PolyPublie
Polytechnique Montréal

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Titre: Title:	Détection de conflits dans les services téléphoniques par la théorie des systèmes à événements discrets
Auteur: Author:	Jounaïdi Ben Hassen
Date:	1999
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Ben Hassen, J. (1999). Détection de conflits dans les services téléphoniques par la théorie des systèmes à événements discrets [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8618/

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/8618/
Directeurs de recherche: Advisors:	
Programme: Program:	Unspecified

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal

https://publications.polymtl.ca

UNIVERSITÉ DE MONTRÉAL

DÉTECTION DE CONFLITS DANS LES SERVICES TÉLÉPHONIQUES PAR LA THÉORIE DES SYSTÈMES À ÉVÉNEMENTS DISCRETS

JOUNAIDI BEN HASSEN
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
JUIN 1999

© Jounaidi Ben Hassen 1999.

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-48836-5

Canada

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

DÉTECTION DE CONFLITS DANS LES SERVICES TÉLÉPHONIQUES PAR LA THÉORIE DES SYSTÈMES À ÉVÉNEMENTS DISCRETS

Présenté par : BEN HASSEN Jounaidi

En vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

A été dûment accepté par le jury d'examen constitué de :

M. MALHAMÉ Roland, PhD., président

M. THISTLE John, PhD., membre et directeur de recherche

Mme. DSSOULI Rachida, PhD., membre

À ma famille...

Remerciements

Je remercie sincèrement tous les membres du jury d'avoir accepté de porter un avis sur ce mémoire.

Je remercie vivement mon directeur de recherche le Professeur John G. Thistle pour m'avoir fourni le support financier nécéssaire pour que je puisse travailler dans les meilleurs conditions. M. John Thistle est à l'origine de cette précieuse occasion pour réaliser ma maîtrise à l'École Polytechnique. Je lui suis très reconnaisant de la confiance qu'il m'a accordé. Je suis aussi très touché de ce que, malgré les tâches multiples qui lui comblent en tant que professeur à l'École Polytechnique, directeur de recherche de plusieurs jeunes chercheurs, M. John Thistle n'a pas cessé de me faire profiter de sa haute compétence, son aide efficace et ses conseils pertinents.

J'adresse mes remerciements aussi au Professeur Roland Malhamé qui a enrichi mon mémoire de ses critiques positives. M. Malhamé m'a fourni aussi un support financier au début de ma maîtrise. Aussi est-ce un grand plaisir pour moi de le remercier aujourd'hui de tout ce qu'il m'a apporté.

C'est le moment aussi pour dire un grand merci à tous ceux qui m'ont aidé pour que je puisse venir au Canada. Je cite entre-autres mon père Baba Hamadi, mon oncle Majid, mes soeurs, ma tante Majida et son époux, mes oncles, sans oublier mon ami Habib et tous les autres. C'est le moment d'exprimer mes plus sincères gratitudes

à toutes ces personnes et de leur dire que vous étiez toujours présents ici avec votre humour, votre charme et votre chaleur. Merci beaucoup pour vous tous et que Dieu vous protège.

Résumé

Le besoin du développement de méthodologie pour traiter les interactions de service devient de plus en plus important dans les systèmes de télécommunications.
Le problème d'interaction de services se manifeste lorsque certains services se comportent de façon incorrecte ou indésirable une fois implantés ensemble. La théorie de commande des systèmes à événements discrets est l'une des méthodes qui s'adressent à une classe d'interactions qui résultent d'un conflit logique entre les services. Parmi ses premiers objectifs est la réduction de la complexité du problème de commande à l'aide d'une synthèse modulaire et une commande décentralisée et hiérarchisée.

Dans ce travail nous formalisons un nouveau générateur pour un petit modèle d'un réseau téléphonique de trois abonnés et des spécifications d'un nombre de services. Dans ce schéma, les services téléphoniques sont implantés au niveau d'un abonné comme des superviseurs modulaires opérant de façon décentralisée, et dont le rôle est de coordoner les actions de l'abonné. Nous analysons les propriétés des langages légaux de ces spécifications en terme de commandabilité et de non blocage, et nous montrons que ces langages implantent des superviseurs propres pour le générateur local de l'abonné. Dans cette théorie, les interactions de services sont considérées comme la vérification d'une propriété de blocage entre langages. Nous utilisons cette caractérisation pour la détection de conflits entre les services étudiés.

Abstract

The need to develop methodologies to manage feature interactions is widely recognized in the telecommunications community. The feature interaction problem occurs when certain services behave in an incorrect or unwanted way once they are put together. Supervisory control theory of discrete event systems is a potiential means of addressing a class of interactions that result from logical conflict among features. Its primary goal is the management of the complexity of control design by means of modular synthesis and decentralized and hierarchical control architectures.

In this thesis we formalize a new generator for a small model of telephone network of three subscribers and specifications of a number of examples of features. In this scheme, service features are implemented at a telephone switch level as modular supervisors operating in a decentralized fashion, whose role is to coordinate the actions of subscribers. We analyse properties of legal languages of these specifications in term of controllability and nonblocking, and we show that these languages correspond to proper supervisors for the local generator model. In the approach based on control theory of discrete event systems, feature interactions are considered as conflicts between marked languages. We use this characterisation for the detection of interactions among the studied services.

Table des matières

Liste des tableaux

Liste des figures

Liste des annexes

Chapitre 1

INTRODUCTION

L'implantation des options téléphoniques est une tâche très importante dans les réseaux et elle est même devenue fondamentale pour répondre aux besoins des consommateurs. Cependant l'implantation de ces options devient de plus en plus une tâche compliquée, vu la complexité de ces services et leur nombre croissant dans les réseaux téléphoniques. En plus, l'existence de plusieurs services au niveau d'un même commutateur pose éventuellement des problèmes d'interactions non désirées entre elles. Ce phénomène est appelé "interférence des services téléphoniques". Nous devons noter que le phénomène d'interférence est un phénomène général pour les systèmes répartis et non pas seulement dans les réseaux téléphoniques. À cause de l'importance de ce problème, plusieurs recherches ont été effectuées pour définir un cadre théorique adéquat pour la modélisation des services et la détection de leurs interférences. Ces études recherchent des solutions théoriques afin d'éviter que les problèmes d'interférences ne se propagent à l'étape d'implémentation, où il serait beaucoup plus difficile de contrôler le fonctionnenemt du réseau.

L'interférence désigne une "opération indésirable" entre un ensemble de règles de

fonctionnement implantées dans un commutateur téléphonique. Cependant, cette définition reste toujours très vague et pose plusieurs questions: Quand dit-on qu'une opération est indésirable, et de quel point de vue est-elle indésirable? Est-ce par rapport aux concepteurs ou par rapport aux consommateurs? Il serait peut-être utile de donner quelques exemples de conflits entre services pour se faire une idée plus claire de ce problème.

Considérons un client A abonné au service CF (Call Forwarding). Ce service lui permet de renvoyer ses appels au numéro d'un autre abonné (C par exemple). Considérons maintenant un autre client B abonné au service $CCBL$ (Continute Composition Busy Line). Ce service permet à son utilisateur de tester automatiquement une ligne téléphonique occupée jusqu'à sa libération. Considérons maintenant la situation suivante: Le service CF de A est actif et tous ses appels sont renvoyés vers C. C est en communication avec un autre abonné quelconque D. Supposons que B essaie de rejoindre A. L'appel de B sera acheminé vers A, puis renvoyé vers C. Puisque la ligne de C est occupée, l'abonné B entendra une tonalité d'occupation. Il va donc activer son service $CCBL$ en attendant que la ligne de A se libère. Ce dernier service va tester l'état de la ligne de A et va trouver qu'elle est libre. Il envoie donc une sonnerie de libération pour demander à B de décrocher. Cependant, lorsque ce dernier décroche l'appel sera de nouveau acheminé vers A puis redirigé vers la ligne de C qui est encore occupée, et de nouveau, B va entendre une tonalité d'occupation. L'abonné B peut activer de nouveau son service, mais il aura toujours le même résultat à moins que la ligne de C se libère. Il est clair que ce conflit est dû, entres autres, au manque de précision dans la documentation des services.

Supposons maintenant que A est abonné au service $CFBL$ (Call Forwarding Busy

Line) au lieu du service CF. Ce service permet de renvoyer les appels de A seulement si la ligne de A est occupée. B est toujours abonné au service $CCBL$. Supposons maintenant que A est en communication avec C, et que B essaie de joindre A. La question qui se pose est la suivante: Est-ce que l'appel de B va être renvoyée à un autre numéro ou est-ce que le service auquel B est abonné va tester continuellement l'état de la ligne de A pour en informer B lorsqu'elle se libère? Dans cet exemple on voit que le conflit est dû au manque de priorité d'un service par rapport à l'autre.

Nous devons mentionner qu'un même service peut provoquer un conflit s'il est activé par plusieurs clients à la fois. Un exemple simple de cette situation est celui de deux clients A et B abonnés au service CF. Si A renvoie ses appels vers B et B renvoie ses appels vers A, aucun d'eux ne pourra recevoir un appel!

L'activation de plusieurs services par le même client, peut provoquer aussi un situation de conflit entre eux. Considérons un abonné A qui est abonné aux deux services $CFBL$ et la messagerie. Ce dernier service permet de renvoyer les appels "sans réponse" (dû à l'absence de l'abonné ou à l'occupation de sa ligne) à un système de traitement de la voix, appelé désormais boîte vocale. Supposons que A est en communication avec un autre abonné B. Si un abonné C tente de joindre A, nous ne pouvons pas déterminer lequel des services va être activé. Est-ce le service $CFBL$, qui va rediriger l'appel de C vers un autre client D, ou est-ce le service de messagerie, qui va déclencher la boîte vocale?

Un grand nombre d'autres exemples d'interférences existent entre les services téléphoniques. Il s'avère donc nécessaire de trouver des méthodes de détection de telles situations lors de la spécification de ces services.

Le présent travail consiste à faire une étude d'un ensemble de trois services télé-

phoniques fourni pour le concours du cinquième atelier international sur les intéractions de services (5th IWFI). Les services étudiés sont:

1. **Conférence à trois** notée TWC (Three Way Calling): ce service permet à un abonné quelconque A d'effectuer deux appels à la fois. Cet abonné initie donc le premier appel avec un autre abonné B, ensuite il le met en attente en pesant la touche *flash* de son appareil téléphonique [1] et compose le numéro d'un autre abonné C. Pour avoir une conversation à trois, l'abonné A pèse de nouveau sur la touche flash.

2. **Appel en attente** notée CW (Call Waiting): ce service permet à un abonné A en conversation téléphonique avec un abonné B, de prendre un nouvel appel provenant d'un autre abonné C. Pour prendre le nouvel appel, l'abonné A pèse sur la touche flash.

3. **Transfert d'appel, ligne occupée** notée CFBL (Call Forwarding, Busy Line): Ce service permet de transférer les appels d'un abonné A vers un autre numéro lorsque sa ligne est occupée.

Après avoir dévéloppé des spécifications de ces services nous avons utilisé une méthode rigoureuse basée sur la théorie de la commande des systèmes à événements discrets pour détecter les interférences qui peuvent se produire à cause de l'implantation simultanée de ces services. Dans la littérature nous trouvons plusieurs approches qui se basent sur cette théorie pour le traitement de conflits entre options téléphoniques. Ce choix se justifie par le fait que cette théorie permet de réduire la complexité du problème par une synthèse modulaire et une commande décentralisée et hiérarchisée.

[1] Certains appareils téléphoniques ne contiennent pas une touche pour flash. Pour générer cet événement, l'abonné doit raccrocher et décrocher rapidement son combiné.

Dans le présent rapport, nous citons au chapitre 2 des travaux de recherche antérieurs pour la détection et/ou la résolution du problème de conflits entre services téléphoniques. À la fin du chapitre nous donnons les travaux basés sur la théorie des systèmes à événements discrets. Au chapitre 3 nous présentons les langages formels ainsi que les éléments de base de la théorie de la commande des systèmes à événements discrets. Au chapitre 4 nous présentons un nouveau générateur d'un petit modèle de réseau téléphonique et des spécifications des trois services "conférence à trois", "appel en attente" et "transfert d'appel, ligne occupée". Nous finissons le chapitre par l'analyse des propriétés de ces spécifications. Au chapitre 5 nous commençons par vérifier l'interférence entre ces services, ensuite nous discutons les résultats obtenus. Nous terminons le rapport par une conclusion au chapitre 6.

Les spécifications présentées et tous les résultats des chapitres 4 et 5 ont été réalisés à l'aide de l'outil logiciel *TCT* [Won88] sous MS-DOS.

Chapitre 2

PRÉSENTATION DES APPROCHES POUR LA DÉTECTION DE CONFLITS

2.1 Le projet P509

Nous présentons dans cette section une approche industrielle pour la détection et la résolution de conflits. Cette approche représente le projet P509 et a été présentée dans [Kim97]. Le projet P509 est un travail coopératif de huit opérateurs de réseaux publiques (PNO): Tele Danmark, Finnet Group, France Telecom, Deutsche Telekom, OTG, KPN, Telephonica et Telia. L'approche proposée par ce projet sépare le traitement des interactions de la création des services. Elle est basée sur le filtrage des services et le traitement de conflits. Le principe de filtrage est introduit pour faciliter la détection des conflits par des méthodes formelles. L'idée est de réduire le nombre de cas à analyser par un filtrage qui permet d'éliminer tous les services qui ne peuvent

pas être en conflit. Le processus de traitement des interactions entre services (SIHP) est un processus séparé responsable de détecter les interactions entre services et de trouver des solutions "optimales" de résolution, mais pas de les implémenter. Ceci signifie que l'approche P509 sépare le SIHP de la création des services. Plusieurs méthodes formelles et informelles à des niveaux d'abstraction différents sont utilisées pour maximiser la détection de conflits. Le SIHP est divisé en cinq sous-processus: préparation, filtrage, détection, résolution, et support. Ce dernier permet la gestion des données utilisées par les quatres autres sous-processus. Pour permettre l'introduction du SIHP dans différentes organisations PNO, P509 propose un modèle d'entreprise simple de l'industrie de télécommunications. Le modèle reflète seulement les rôles les plus importants dans la création, intégration, fourniture, et l'usage des services de télécommunications. Le traitement des interactions est considéré dans cette approche comme une partie de l'intégration de service. Ainsi, le modèle d'entreprise inclut deux nouveaux rôles: L'intégrateur de service et le médiateur du service.

1. L'intégrateur de service a pour rôle l'intégration des services créés par les différents créateurs de services au niveau d'un seul fournisseur de service. Il est le responsable de la détection et de la résolution des interactions entre services fournies par le même fournisseur de service, dites les interactions *intra-portfolio.*

2. Le médiateur de service a pour rôle l'intégration des services offerts par des différents fournisseurs de services. Il est responsable de la détection et de la résolution des interactions entre services fournies par plusieurs fournisseurs de services, dites les interactions *inter-portfolio.*

Une approche qui a été développée dans le cadre du projet P509 a été présentée dans [AC97]. Cette approche combine la détection et la résolution du conflit dans

l'environnement de création du service (Service Creation Environment (SCE)) et dans l'environnement d'exécution du service (Service Execution Environmnent (SEE)). Dans le SCE, les services sont spécifiés de façon formelle, puis testés par des "observateurs passifs". Cette opération consiste à faire la simulation exhaustive d'un modèle exécutable. Ces observateurs décrivent les spécifications du services par des automates exécutés ensembles. Ils sont appelés passifs car ils ne peuvent pas modifier les fonctionnalités des services. La méthode de vérification est basée sur la technique d'observation d'un modèle SDL durant la simulation, où les observateurs sont traités comme des modules externes associés au système SDL. Les "observateurs actifs" sont introduits dans le SEE. Ils permettent ou bien la détection des interactions qui n'ont pas été détectées dans le SCE par les observateurs passifs, ou bien la séquence d'exécution qui a amené à une interaction détectée dans le SCE. Contrairement aux observateurs passifs, les observateurs actifs peuvent modifier les fonctionnalités du système en temps réel pour résoudre le conflit.

2.2 Détection des interactions par les spécifications rationnelles et formelles

Nous citons à titre d'exemples deux méthodes qui ont été présentées dans [Joh97] et [FMD97]. Dans ces méthodes, le réseau téléphonique est considéré comme un système qui réagit à des entrées (appelées les stimuli du système), pour produire des sorties.

Dans [Joh97], nous trouvons une présentation d'un outil de travail, appelé Tussilago, pour la spécification des services téléphoniques. L'approche est basée sur la spécification de chaque service par un module séparé. La structuration des spécifications formelles en modules permet une plus grande réutilisabilité de ces spécifications.

En plus, ceci permet l'addition de nouveaux modules afin d'avoir plus de fonctionnalités dans le système. Les stimuli sont modélisés par des événements externes. Le système réagit à ces événements par un changement d'état. Pour cela on définit trois ensembles de variables: l'ensemble des abonnés, l'ensemble des états, et l'ensemble des événements. La sémantique du système est donnée sous forme de règles composées de formules atomiques, de préconditions et de postconditions. Les règles qui définissent un service sont regroupées dans un module. Chaque module doit être capable d'inclure de nouveaux événements. La spécification entière est obtenue par la composition de ces modules. Dans [Joh97], nous trouvons un exemple de spécification du service CFBL.

Une autre approche est présentée dans [FMD97]. Cette approche utilise la condition de cohérence pour la détection d'interférences entre service. Dans [DBS+95, BEM92], un ordre de raffinement est défini entre les spécifications rationnelles. Il a été démontré que chaque paire de relations rationnelles possède une borne supérieure, et sous une certaine condition une borne inférieure. Cette condition est appelée *la condition de consistance.* Un système téléphonique est considéré comme un système dynamique c'est-à-dire que les sorties du système ne dépend pas seulement de l'entrée courante (le stimulus), mais de toute l'historique des entrées (les stimuli). La spécification d'un système dynamique est définie par l'espace des entrées, l'espace des sorties, et la relation entrée-sortie. Le comportement du système est donné par des règles d'induction, basées sur la logique de premier ordre. En utilisant cette logique, une définition inductive d'une relation est donnée par un ensemble d'axiomes de la forme

$$(\bigwedge_{i=1}^{n} A_i) \Rightarrow B, n \in N$$

Les entrées du système sont fournies par les ingénieurs du réseau téléphonique (abonner, désabonner un client), et les appareils téléphoniques (PickUp, HangUp, Dial). Les sorties du système sont fournies aux appareils téléphoniques (Ring, Connect, Disconnect..). Dans cette approche, le nombre d'utilisateurs et le nombre de connexions sont illimités. L'espace des entrées est défini par

$$I \hat{=} \{PkUp\} \times PhoneId \cup \{HgUp\} \times PhoneId \cap \{Dial\} \times PhoneId \times PhoneId$$

Où $PhoneId$ désigne le numéro de téléphone de l'abonné et supposé unique. L'espace des sorties est défini par

$$O \hat{=} \mathcal{P} \left(\begin{array}{ll} \{DiTo\} \times PhoneId & \cup \\ \{Ring\} \times PhoneId \times PhoneId & \cup \\ \{BuTo\} \times PhoneId & \cup \\ \{Conn\} \times PhoneId \times PhoneId & \cup \\ \{Disc\} \times PhoneId \times PhoneId & \end{array} \right)$$

où

- $DiTo$ est la tonalité de composition.
- $Ring$ reprérente la sonnerie d'appel.
- $BuTo$ désigne une tonalité d'occupation.
- $Conn$ est l'établissement d'une connexion, et
- $Disc$ une déconnexion.

Ainsi, une sortie est un ensemble de commandes. Pour le service téléphonique standard (POTS), la sortie est toujours un singleton, mais pour les autres services comme

TWC ou CW, la sortie peut être un ensemble de plusieurs éléments.

2.3 Technique de spécification multiparadigme

Nous présentons dans cette section une approche basée sur une technique de spécification multiparadigme développée par P. Zave dans [JZ96, Zav98]. Dans cette technique les entrées du système prennent la forme d'événements atomiques et séquentiels et les sorties la forme de changement d'états. Une spécification multiparadigme consiste en un ensemble de spécifications partielles. Une de ces spécifications est écrite en un langage orienté-modèle ou orienté-état tel que VDM ou Z. Cette spécification est dite spécification "centrale". Les autres spécifications partielles sont écrites en d'autres langages où les représentations d'états sont plus restrictives tels que les automates finis et la logique de premier ordre. Ces spécifications forme ensembles la spécification "périphérique". Chaque spécification partielle est équivalente à une théorie dans la logique de prédicat de premier ordre. Un "phénomène du domaine" est formulé comme un prédicat dont les arguments sont des éléments d'un univers d'individus distincts. Le vocabulaire d'une spécification partielle est l'ensemble des prédicats apparaissant dans sa théorie. Les éléments de l'univers incluent les événements d'entrées. Ces événements sont partitionnés en une collection fini de types d'événements. Chaque type d'événements est un "phénomène du domaine" et est formulé comme un prédicat unaire. Une classe d'événements est un ensemble d'événements caractérisé par un prédicat unaire défini sur les événements. Chaque spécification partielle (ou au moins sa partie de sémantique qui interagit avec les autres spécifications partielles) est équivalente à une assertion dans la logique de prédicat de premier ordre. Dans [JZ98] une architecture virtuelle de composition de services distribués (Distributed Feature

Composition (DFC) virtual architecture) est introduite. Cette architecture fournit une nouvelle fondation pour la description modulaire des services téléphoniques, ainsi qu'une structure sémantique qui permet le traitement et l'élimination des interactions indésirables. Cette architecture virtuelle est utilisée dans [Zav98] pour le traitement sémantique d'interactions dans la classe de services "couverture d'appel" ("call coverage" features), ce qui revient en fait à une vérification formelle des propriétés du système.

2.4 Méthodes orientées-objets pour la détection de conflit

Dans la littérature nous trouvons plusieurs méthodes orientées-objets pour la détection et la résolution d'interférences entre services téléphoniques [IE97, Pre97]. L'approche décrite au [IE97] est basée sur les étapes suivantes:

1. Spécifier formellement chaque service impliqué dans l'interaction.
2. Faire une composition parallèle des spécification des deux services analysés. Ceci correspond à faire un produit d'automates qui spécifie les comportements des objets de chaque spécification.
3. Identifier les états de conflits en analysant l'automate produit dans l'étape 2.

Une interaction correspond à un état de conflit. Cet état peut être soit un état où le même événement amène à deux états différents (non-déterminisme), un état de verrou mortel à partir duquel aucune transition ne peut être exécutée, ou une violation de contraintes liée à un état conséquent; c'est le cas d'un état créé par le produit mais qui résulte de deux états incompatibles.

La méthode de résolution de conflits est formée de trois stratégies. L'une de ces stratégies est choisie selon le type d'interaction. Les stratégies utilisées sont les suivantes:

1. Faire une composition par le choix exclusif des deux spécifications impliquées dans une interaction.

2. Résoudre l'interaction en permettant à un service de cacher certains événements de l'autre service.

3. Établir un protocole entre les services impliqués dans une interaction. Ce protocole consiste en un échange d'informations. Cette approche est plus adaptée au cas où les services sont implantés sur des sites différents.

Une autre approche a été donnée dans [Pre97], où le problème de conflit est traité dans un nouveau style de programmation dit orienté-services (FOP). En s'inspirant de la programmation orientée-objets, la FOP permet la composition des services sous formes d'objets, où un ensemble de services remplace la structure conventionnelle des classes. Pour construire un objet (ou un type de classe), les services sont ajoutés l'un après l'autre dans un ordre particulier. En plus des avantages de la programmation orientée-objets, ce style permet une plus grande clarification des dépendances entre les services.

2.5 Méthodes basées sur LOTOS pour la détection de conflits

La technique de description formelle LOTOS (Language of Temporal Ordering Specification) a été utilisée par plusieurs auteurs pour la spécification des services et la

détection de leurs conflits. Luigi Logrippo et ses étudiants proposent dans [KL98] un modèle de LOTOS pour la spécification d'un modèle d'appel dans un réseau intelligent. Le but derrière l'utilisation de LOTOS est d'avoir une spécification ou un modèle exécutable qui peut être utilisé pour la spécification, la validation et la détection des interactions. Chaque nouveau service doit être spécifié indépendamment des autres et ajouté à la spécification globale. Dans cette approche, il y a interaction si certaines propriétés ne sont pas satisfaites dans la spécification formelle.

LOTOS est aussi utilisé dans l'approche proposée par Gibson dans [Gib97, Gib98]. Cette approche est basée sur le développement de deux modèles: un modèle exécutable (écrit en LOTOS) et un modèle logique. Le premier modèle est basé sur la définition d'un ensemble d'objets qui servent pour la validation du comportement dynamique du système. Le deuxième modèle est basé sur la définition d'un ensemble de propriétés qui peuvent être validées statiquement. L'approche proposée dans [Tho97] utilise aussi LOTOS pour le développement d'un modèle haut niveau du réseau. LOTOS est utilisé dans cette approche comme un langage de représentation. D'autres outils comme LOLA (LOTOS Laboratory) sont utilisés pour faire des simulations du système. Le modèle haut niveau est motivé par la nécessité de spécifier les services de manière qu'ils soient indépendants entre eux, et de réutiliser le comportement existant le plus possible. Par exemple, si un service affecte seulement le comportement de la partie appelante, alors il serait utile dans la spécification du service de ne pas redéfinir le comportement de la partie destinataire de l'appel.

Le projet ANISE (Architectural Notions In Service Engineering) [Tur98, Tur97] se base aussi sur la validation des services à l'aide de LOTOS. ANISE est un langage pour la définition et l'analyse des services téléphoniques. En plus, il fournit un niveau plus raffiné de détails des services. Les services sont implantés comme des comportements

élémentaires du système. Le comportement global est considéré comme la conjonction de tous les comportements élémentaires. Dans cette approche, POTS est traité aussi comme un comportement élémentaire, donc comme un service séparé. La validation des services se fait en translatant automatiquement à LOTOS les scénarios d'appels possibles, ce qui permet à un service d'être validé soit séparément, soit en combinaison avec d'autres services.

2.6 Détection et résolution de conflits par les systèmes à événements discrets

Le problème d'interactions de services a été traité pour la première fois dans le cadre de la théorie de commande par Thistle dans [THM93]. Depuis, plusieurs travaux ont été réalisés se basant sur cette théorie. Dans [TMH97] les interactions de services téléphoniques sont interprétées comme le résultat de l'implantation modulaire et décentralisée des services dans les systèmes distribués. Le système est modélisé par un générateur de langage formel sur un alphabet d'événements. Les mots du langage représentent les séquences d'événements qui peuvent se produire dans le système. Les mots qui définissent des tâches "achevées" (appelés mots marqués) forment le langage marqué (ou but). Les superviseurs commandent les générateurs par l'interdiction de l'occurrence de certains événements en se basant sur l'historique de la séquence d'événements observables. [Thi96, TMHL97] donnent plus de détails sur la notion de supervision dans les systèmes à événements discrets. Les superviseurs doivent être non bloquants, c'est-à-dire que tout mot engendré sous la supervision du superviseur peut être étendu en un mot marqué. À un commutateur donné, chaque service est assuré par un superviseur implanté dans le commutateur. Le problème d'interférence

revient donc à un problème de conflit entre plusieurs superviseurs modulaires implantés au niveau d'un même commutateur local. Pour résoudre ce conflit, à chaque service on affecte un degré de priorité. Ainsi, si un événement généré est interprété de deux façons différentes par deux superviseurs, alors cet événement doit être invisible (ou caché) par rapport au superviseur du service le moins prioritaire. La formulation du problème consiste en la définition d'une application $\theta : L \longrightarrow \Sigma^*$, dite *reporter map* dont le rôle est de masquer l'occurrence de certains événements du superviseur le moins prioritaire:

$$\begin{aligned} \theta(1_\Sigma) &= 1_\Sigma \\ \theta(s\sigma) &= \begin{cases} \theta(s)\sigma \\ \theta(s) \text{ pour certains } \sigma \in \Sigma \end{cases} \end{aligned}$$

avec Σ l'ensemble d'événements du système, L un langage préfix-fermé, 1_Σ est appelé le mot vide (voir 3.1), $s \in L$ et $\sigma \in \Sigma$. Cette approche utilise une décomposition hiérarchisée du système, où le bas niveau est représenté par le système modélisé et les superviseurs implantés, le haut niveau est le résultat de l'application de la fonction θ. Cette fonction doit vérifier certaines propriétés de cohérence et d'observabilité. Dans [ZW90] nous trouvons plus de détails sur la décomposition hiérarchique dans les systèmes à événements discrets. Parmi les travaux qui ont traité le conflit des services téléphoniques en se basant sur cette approche nous citons [Mun98] dans lequel des spécifications des deux services CSMI (Call Screening, Monitor and Intercept) et CW(Call Waiting) ont été dévéloppées. La méthode pour la résolution de conflit entre ces deux services est aussi présentée.

Yi-Liang Chen et Stéphane Lafortune présentent dans [CLL96, CLL97] une approche complémentaire à l'approche dévéloppée par Thistle et al. dans [TMH97] et

[TMHL97]. Cette méthode se base sur la définition d'une nouvelle approche de commande modulaire qui peut servir comme outil pour la détection et la résolution de conflits. L'idée est de définir un mécanisme de priorité basé sur des fonctions de priorités des superviseurs individuels qui implantent les différents services. Pour cette raison, ce schéma est appelé commande modulaire avec priorités (CMP). Les conflits peuvent ainsi être résolus en forçant/interdisant certaines actions de commande lorsque ceci est nécessaire en se basant sur les fonctions de priorités (voir [CLL97]). Contrairement à l'approche citée dans [TMH97], les superviseurs dans le schéma CMP peuvent toujours observer les mêmes séquences d'événements.

2.7 Conclusion

Dans ce chapitre nous avons présenté plusieurs méthodes pour la détection et la résolution de conflits. Il est clair que plusieurs approches sont adaptées pour ce problème. Cependant, nous pouvons les grouper en deux grandes classes: les méthodes formelles et les méthodes informelles. Ceux qui utilisent des méthodes formelles expliquent leur choix par le fait que (voir [TMH97, Kim97]):

- Ces méthodes donnent des solutions indépendantes de l'architecture ou du type du système.
- Ces méthodes permettent de traiter le problème de conflits à une étape avancée de l'implémentation des services.
- Étant donné que ces méthodes se basent sur des résultats théoriques, elles permettent de trouver des solutions rigoureuses.
- À cause de la complexité du problème traité, il est nécéssaire de le traiter dans

un cadre formel.

- Ces méthodes donnent des solutions théoriques indépendantes d'un service particulier, ce qui permet une plus grande utilisabilité.

Ceux qui utilisent des méthodes informelles expliquent leur choix par le fait que (voir [Kim97]):

- Les méthodes formelles sont généralement complexes, et coûteuses en terme de temps d'exécution.
- Étant donné que les méthodes formelles se basent sur des résultats théoriques, elles ne peuvent être utilisées que par des gens spécialistes et compétents.
- La validation des propriétés formelles ne peut être faite que par des outils adéquats.

Chapitre 3

LA COMMANDE DES SYSTÈMES À ÉVÉNEMENTS DISCRETS

Un système à événements discrets (SED) est un système dynamique qui évolue en fonction de l'occurrence d'événements instantanés. Dans ce chapitre, nous faisons un survol des aspects logiques de tels systèmes. La théorie que nous présentons a été introduite par Ramadge et Wonham qui ont enrichi ces systèmes par une notion de commande, basée sur la théorie des langages formels (voir [RW89] pour un aperçu).

3.1 Langages formels

Un *alphabet* Σ est un ensemble fini et non vide de symboles appelés des *lettres*. Une suite de lettres

$$m = \sigma_1\sigma_2...\sigma_i...\sigma_n, (\sigma_i \in \Sigma, i \in \{1, ..., n\})$$

est appelée un *mot* sur l'alphabet Σ. Σ^+ désigne l'ensemble de tous les mots sur Σ.

La *concaténation* de deux mots $u, v \in \Sigma^+$ est le mot $u.v$, c'est à dire la suite de

symboles u suivie de la suite v. $u.v$ est souvent noté uv. Il est utile d'introduire le *mot vide* sur Σ, noté 1_Σ, et qui sert comme élément neutre pour la concaténation. On désigne par Σ^* la réunion $\Sigma^+ \cup 1_\Sigma$. Il est à noter que la structure algébrique $(\Sigma^*, .)$, où $(.)$ désigne l'opérateur de concaténation, est un *monoïde*.

Définition 3.1 *Un mot v est un facteur d'un mot u s'il existe deux mots u_1, u_2 tel que $u = u_1 v u_2$. Si $u_1 = 1_\Sigma$ (resp. $u_2 = 1_\Sigma$), alors v est un facteur gauche ou préfixe (resp. facteur droite ou suffixe) de u.*

Un *langage* sur Σ est un sous-ensemble de Σ^* (pour avoir plus de détails, nous conseillons au lecteur de voir [HU69]).

Le produit de concaténation induit un produit de langages:

$$L_1 L_2 := \{uv \in \Sigma^* : u \in L_1, v \in L_2\},$$

où $L_1, L_2 \subseteq \Sigma^*$. Il est à noter que ce produit de langages est associatif:

$$(L_1 L_2) L_3 = L_1 (L_2 L_3).$$

Définition 3.2 (Fermeture de Kleene) *La fermeture de Kleene d'un langage $L \subseteq \Sigma^*$, noté L^* est définie par récurrence comme suit:*

$$L^* := \cup_{n \geq 0} L^n,$$

avec:

$L^0 := \{1_\Sigma\}$ *et,*

$L^{i+1} := L^i L$ *pour tout* $i \geq 0$.

Définition 3.3 *Le résiduel $u^{-1}L$ d'un langage $L \subseteq \Sigma^*$ par rapport à un mot $u \in \Sigma^*$ est défini par:*

$$u^{-1}L := \{v \in \Sigma^* : uv \in L\}.$$

Les langages formels sont souvent représentés sous forme de machines. La notion de *résiduel* d'un langage est lié à celle de l'état d'une machine correspondante.

3.2 Les automates finis

3.2.1 Définitions de base

Un automate fini est un quintuplet $A = (\Sigma, X, \delta, x_0, T)$, où:

- Σ est un alphabet fini.
- X est un ensemble fini d'états.
- $\delta : \Sigma \times X \longrightarrow 2^X$ est une fonction (partielle) de transition.
- $x_0 \in X$ est l'état initial, et
- $T \subseteq X$ est un ensemble d'états terminaux.

La notation $\delta(\sigma, x)!$ signifie que $\delta(\sigma, x)$ existe.

Si $(\forall \sigma \in \Sigma \ \forall x \in X)$, $[\delta(\sigma, x) \leq 1]$, alors A est dit déterministe. Dans ce cas, la fonction δ est définie comme suit:

$$\delta : \Sigma \times X \longrightarrow X.$$

Toutefois, il est important de noter que tout automate $A = (\Sigma, X, \delta, x_0, T)$ non déterministe est équivalent à un autre automate déterministe B qui reconnait le même langage, avec:

$$B = (\Sigma, 2^X, \gamma, \{x_0\}, \{X' \subseteq X : X' \cap T \neq \emptyset\})$$

où:

$\gamma : \Sigma \times 2^X \longrightarrow 2^X$

$(\sigma, X') \mapsto \bigcup_{x' \in X'} \delta(\sigma, x')$.

Dans tout ce qui suit A est considéré comme un automate déterministe.

La fonction δ est souvent étendue comme suit:

$$\begin{array}{rccl} \hat{\delta}: & \Sigma^* \times X & \longrightarrow & X \\ & (1_\Sigma, x) & \mapsto & \{x\} \\ & (s\sigma, x) & \mapsto & \delta(\sigma, \hat{\delta}(s, x)). \end{array}$$

Le langage reconnu (dit aussi achevé ou accepté) par A est:

$$L(A) := \{m \in \Sigma^* : \hat{\delta}(m, x_0) \in T\}$$

En d'autre termes, le langage achevé est l'ensemble des mots qui représentent des tâches finies (ou complètes).

3.2.2 Opérations sur les automates

Soit $A = (\Sigma, X, \delta, x_0, T)$ un automate fini. Le composant accessible de A est:

$$A_c(A) = (\Sigma, X_{ac}, \delta_{ac}, q_0, T_{ac})$$

avec:

$X_{ac} = \{x \in X : (\exists m \in \Sigma^*)[x \in \delta(m, x_0)]\}$;

$\delta_{ac} = \delta \uparrow (\Sigma \times X_{ac})$(c'est-à-dire la restriction de δ à $(\Sigma \times X_{ac})$);

et $T_{ac} = T \cap X_{ac}$.

Il est évident que $L(A) = L(A_c(A))$.

A est dit accessible si:

$$A = A_c(A).$$

A est dit co-accessible si:

$$(\forall m \in \Sigma^*)[\delta(m, x_0)! \Longrightarrow (\exists m' \in \Sigma^*)[\delta(mm', x_0) \in T]].$$

Soit $A_t = (\Sigma, X \dot{\cup} \{x_t\}, \delta_t, x_0, T)$ défini par:

$$\begin{array}{rccl} \delta_t : & \Sigma \times X \dot{\cup} \{x_t\} & \longrightarrow & X \dot{\cup} \{x_t\} \\ & (\sigma, x) & \mapsto & \delta(\sigma, x), \forall x \in X, \delta(\sigma, x)! \\ & (\sigma, x) & \mapsto & \{x_t\} \text{ autrement.} \end{array}$$

La fonction de transition δ_t est totale. En conséquence, A_t est dit total. En plus $L(A_t) = L(A)$.

Soit $A = (\Sigma, X, \delta, x_0, T)$ un automate déterministe et total. Le complément A^c de A est donné par:

$$A^c = (\Sigma, X, \delta, x_0, X \setminus T).$$

Et en plus:

$$L(A^c) = \Sigma^* \setminus L(A).$$

Soit $A_1 = (\Sigma, X_1, \delta_1, x_{0_1}, T_1)$ et $A_2 = (\Sigma, X_2, \delta_2, x_{0_2}, T_2)$ deux automates finis définis sur le même alphabet. Le produit de A_1 et A_2 est l'automate fini

$$A_1 \times A_2 = (\Sigma, X_1 \times X_2, \delta_1 \times \delta_2, (x_{0_1}, x_{0_1}), T_1 \times T_2),$$

où:

$$\begin{aligned} \delta_1 \times \delta_2 : \ \Sigma \times (X_1 \times X_2) &\longrightarrow X_1 \times X_2 \\ (\sigma, (x_1, x_2)) &\mapsto (x_1', x_2') \in X_1 \times X_2 : x_1' = \delta_1(\sigma, x_1), x_2' = \delta_2(\sigma, x_2) \end{aligned}$$

$A_1 \times A_2$ reconnait $L(A_1) \cap L(A_2)$.

Soit $A_1 = (\Sigma, X_1, \delta_1, x_{0_1}, T_1)$ et $A_2 = (\Sigma, X_2, \delta_2, x_{0_2}, T_2)$ deux automates finis. La réunion de A_1 et A_2 est l'automate fini

$$A_1 \cup A_2 = (\Sigma, \{x_0\} \cup X_1 \cup X_2, \delta_u, x_0, T_u)$$

où:

$$\begin{aligned} \delta_u : \ \Sigma \times \{x_0\} \cup X_1 \cup X_2 &\longrightarrow \{x_0\} \cup X_1 \cup X_2 \\ (\sigma, x_0) &\mapsto \delta_1(\sigma, x_{0_1}) \cup \delta_2(\sigma, x_{0_2}) \\ (\sigma, x_1) &\mapsto \delta_1(\sigma, x_1), \forall x_1 \in X_1 \\ (\sigma, x_2) &\mapsto \delta_2(\sigma, x_2), \forall x_2 \in X_2 \end{aligned}$$

$T_u = T_1 \cup T_2$ si ni x_{0_1} ni x_{0_2} est un état marqué.

$T_u = T_1 \cup T_2 \cup \{x_0\}$, autrement.

T_u n'est pas forcément déterministe.

Un langage reconnu par un automate fini est dit *reconnaissable.*

Théorème 3.1 (Myhill-Nerode) *Soit L un langage reconnaissable. Alors, il existe un automate fini et déterministe $A = (\Sigma, X, \delta, x_0, T)$ tel que $L(A) = L$, et pour tout automate fini et déterministe A' qui reconnaît L, A ne contient pas plus d'états que A'. De plus cet automate est unique à un isomorphisme près. Dans ce cas, A est dit l'automate minimal qui reconnaît L.*

Les *expressions rationnelles* sur Σ et les langages qu'elles désignent sont définies de façon récursive comme suit:

1. $\emptyset$ est une expression rationnelle qui désigne le langage vide.

2. Si $m \in \Sigma^*$ alors m est une expression rationnelle qui désigne le langage $\{m\}$.

3. Soit r et s deux expressions rationnelles qui désigne $R, S \subseteq \Sigma^*$ respectivement. Alors $(r + s)$, (rs) et (r^*) sont des expressions rationnelles qui désignent $R \cup S$, RS et R^* respectivement.

Nous omettons les parenthèses selon les règles de précédence suivantes: $*$ précède le produit et le produit précède $+$.
Un langage qui est désigné par une expression rationnelle est dit *rationnel.*

Théorème 3.2 (Kleene) *Un langage est reconnaissable si et seulement si il est rationnel.*

3.3 Cadre formel de la théorie R-W

3.3.1 Générateurs

Dans un modèle logique des systèmes à événements discrets, nous ne nous intéressons qu'aux séquences d'événements que le système génère et non pas à une modélisation explicite du temps ou d'aspects stochastiques. Pour cela, les systèmes à événements discrets sont modélisés sous forme de *générateurs* de langages formels:

$$G = (\Sigma, Q, \delta, q_0, Q_b)$$

où:

- Σ est un alphabet (représentant les événements).
- Q est un ensemble d'états.
- $\delta : \Sigma \times Q \longrightarrow Q$ est une fonction de transition (partielle).
- $q_0 \in Q$ est l'état initial; et
- $Q_b \subseteq Q$ est un ensemble d'états dits états buts ou marqués.

Comme d'habitude, $\delta(\sigma, x)!$ signifie que $\delta(\sigma, x)$ est définie, et la fonction δ est étendue de la même façon que dans le cas des automates.

Le générateur est donc une machine qui démarre à l'état 0 et génère des mots en exécutant les transitions qui sont définies par sa fonction de transition partielle δ. Le sous-ensemble de mots $s \in \Sigma^*$ tel que $\delta(\sigma, x)!$ représente le *langage engendré* par G est défini par:

$$L(G) := \{m \in \Sigma^* : \delta(m, x)!\}.$$

Le *langage achevé* de G est donné par:

$$L_b(G) := \{m \in \Sigma^* : \delta(m, q_0) \in Q_b\} \subseteq L(G).$$

Donc, si un générateur G est représenté par un automate (déterministe) A, le langage achevé de G sera le langage reconnu par A:

$$L_b(G) = L(A).$$

Pour tout langage $L \subseteq \Sigma^*$, $\overline{L}$ désigne la fermeture (sous préfixes) de L, soit:

$$\overline{L} := \{m \in \Sigma^* : \exists n \in \Sigma^*, mn \in L\},$$

c'est à dire l'ensemble des préfixes de L.

Soient $L, M \subseteq \Sigma^*$, alors:

1. Si $L = \overline{L}$, alors L est dit *fermé*.
2. Si $L = \overline{L} \cap M$, alors L est dit *fermé par rapport à* M.
3. Si $M \supseteq \overline{M} \cap L$, alors M est dit *L-fermé*.

Il est évident que $L(G)$ est fermé par définition.

On dit que G est non bloquant si $L(G) = \overline{L_b(G)}$. De façon informelle, cette définition veut dire que tout mot engendré par G peut être étendu de façon à obtenir un mot achevé de G.

L'une des opérations les plus utiles dans la modélisation des SED est le *produit synchrone* défini à partir des *projections naturelles.*

Définition 3.4 (Projection naturelle) *Soient* $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$ *et* $\Sigma^* = \Sigma_1^* \cup \Sigma_2^*$. *La projection naturelle* P_i *(pour* $i = 1, 2$*) est définie par:*

$$\begin{aligned} P_i: \ \Sigma^* &\longrightarrow \Sigma_i^*, i = 1, 2 \\ 1_\Sigma &\mapsto 1_{\Sigma_i} \\ \sigma &\mapsto \sigma \ si \ \sigma \in \Sigma_i \\ \sigma &\mapsto 1_{\Sigma_i} \ si \ \sigma \notin \Sigma_i \\ s\sigma &\mapsto P_i(s)P_i(\sigma) \end{aligned}$$

Informellement, l'effet de P_i sur un mot s est simplement d'effacer tous les symboles qui n'appartiennent pas à Σ_i.

La projection naturelle est étendue sur les langages comme suit:

$$\begin{aligned} P_i: \ \mathcal{P}(\Sigma^*) &\longrightarrow \mathcal{P}(\Sigma_i^*) \\ K &\mapsto \{P_i(s) : s \in K\}. \end{aligned}$$

$P_i^{-1} : \mathcal{P}(\Sigma_i^*) \longrightarrow \mathcal{P}(\Sigma^*)$ est définie comme l'inverse de la fonction P_i

$$P_i^{-1} : M \mapsto \{s \in \Sigma^* : P_i(s) \in M\}.$$

Définition 3.5 *Le produit synchrone de* L_1 *et* L_2 *est défini par:*

$$L_1 \| L_2 := P_1^{-1} L_1 \cap P_2^{-1} L_2,$$

où

$$P_i^{-1}L_i := \{s \in \Sigma^* : P_i(s) \in L_i\}.$$

Les deux propriétés fondamentales du produit synchrone sont les suivantes:

1. Le produit synchrone est associatif: $(L_1 \| L_2) \| L_3 = L_1 \| (L_2 \| L_3)$.

2. Soit G_1, G_2 deux générateurs non bloquants. Le générateur G tel que $L(G) = L(G_1) \| L(G_2)$ et $L_b(G) = L_b(G_1) \| L_b(G_2)$ n'est pas forcément non bloquant.

Pour ajouter un mécanisme de commande, nous supposons que certains événements du système peuvent être empêchés de se produire. Ceci nous permet d'influencer l'évolution du système en interdisant l'occurrence de certains événements à des instants précis. Pour modéliser un tel mécanisme de commande, nous supposons que l'alphabet Σ est divisé en deux parties disjointes:

1. Σ_c, qui désigne l'ensemble d'événements commandables, et

2. Σ_u, qui désigne l'ensemble d'événements non commandables.

Les événements commandables seront interprétés comme étant succeptibles d'être empêchés par un contrôleur ou superviseur.

3.3.2 Superviseurs

Le rôle d'un superviseur est de restreindre le langage du générateur (en empêchant certains événements commandables) de façon à le garder à l'intérieur de limites acceptées. Lors de la conception d'un système, quelques contraintes de fonctionnement doivent être respectées. Pour respecter ces dernières, un superviseur approprié doit

être alors synthétisé. Typiquement, nous souhaitons toujours qu'il soit le moins restrictif possible, tout en respectant les contraintes imposées.

Soit $G = (\Sigma, Q, \delta, q_0, Q_b)$ un générateur, où $\Sigma = \Sigma_u \dot{\cup} \Sigma_c$. L'ensemble d'entrées de commande est donné par:

$$C := \{\Gamma \subseteq \Sigma : \Gamma \supseteq \Sigma_u\}.$$

Un superviseur pour G est une fonction définie comme suit:

$$V : L(G) \longrightarrow C$$

Nous désignons par V/G le générateur G sous la supervision de V. Le langage $L(V/G)$ engendré par V/G (G sous la supervision de V) est défini par une récurrence sur la longueur des mots comme suit:

1. $1_\Sigma \in L(V/G)$

2. $(\forall s \in \Sigma^*, \forall \sigma \in \Sigma)(s \in L(V/G) \wedge \sigma \in V(s) \wedge s\sigma \in L(G) \Longleftrightarrow s\sigma \in L(V/G))$.

$L(V/G)$ est le plus petit langage qui vérifie ces deux dernières propriétés. Le langage achevé $L_b(V/G)$ est défini par:

$$L_b(V/G) := L(V/G) \cap L_b(G)$$

Donc, c'est l'ensemble de mots de G qui survivent sous la supervision de V.

Définition 3.6 *Le superviseur V est dit non bloquant pour G si et seulement si:*

$$L(V/G) = \overline{L_b(V/G)} = \overline{L(V/G) \cap L_b(G)}.$$

De façon informelle, cette définition signifie que tout mot engendré par le générateur G sous la supervision de V, peut être étendu en un mot achevé de V/G.

3.4 Supervision monolithique

3.4.1 Commandabilité

Définition 3.7 *Soit $G = (\Sigma, Q, \delta, q_0, Q_b)$ un générateur et soit $K \in \Sigma^*$. Le langage K est dit commandable par rapport à G si et seulement si:*

$$\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}.$$

En d'autres termes, cette définition veut dire que l'appartenance à la fermeture de K est invariante sous la génération d'événements non commandables.

Théorème 3.3 *Soit $K \subseteq L_b(G)$ un langage non vide. Alors, il existe un superviseur V non bloquant pour G tel que $L_b(V/G) = K$ si et seulement si:*

1. *K est commandable par rapport à G.*
2. *K est fermé par rapport à $L_b(G)$.*

Corollaire 3.1 *Soit $K \subseteq L(G)$ un langage non vide et fermé. Alors, il existe un superviseur V pour G tel que $L(V/G) = K$ si et seulement si K est commandable par rapport à G.*

Soit $G = (\Sigma, Q, \delta, q_0, Q_b)$ un générateur et soit $E \subseteq \Sigma^*$. Dans la pratique, nous choisissons $E \subseteq L_b(G)$. E (appelé *langage légal*) désigne en fait la formalisation des

contraintes imposées par le système.

Il est important de définir la classe de sous langages commandables de E:

$$\mathcal{C}(E) = \{K \subseteq E : K \text{ est commandable par rapport à } G\}.$$

$\mathcal{C}(E)$ est un ensemble ordonné. En plus il est un sup-demi-treillis.

Proposition 3.1 *$\mathcal{C}(E)$ est non vide, et fermé sous réunions arbitraires. En particulier, $\mathcal{C}(E)$ contient sa propre borne supérieure (par rapport à $\subseteq$), désigné par $sup\mathcal{C}(E)$.*

Proposition 3.2 *Soit $E \subseteq \Sigma^*$ un langage $L_b(G)$-fermé. Alors, $sup\mathcal{C}(E \cap L_b(G))$ est fermé par rapport à $L_b(G)$.*

Théorème 3.4 *Soit $E \subseteq \Sigma^*$ un langage $L_b(G)$-fermé.*
Soit $K = sup\mathcal{C}(E \cap L_b(G))$.
Si $K \neq \emptyset$, alors il existe un superviseur non bloquant V tel que

$$L_b(V/G) = K.$$

Donc, ce dernier résultat nous assure l'existence d'un superviseur non bloquant qui soit le moins restrictif possible si tout mot achevé s de G qui est un préfixe du langage E est un mot de E.

Soit $G = (\Sigma, Q, \delta, q_0, Q_b)$ un générateur, $K \subseteq \Sigma^*$ et V un superviseur non bloquant pour G tel que:

1. $L_b(V/G) = K$,

2. $L(V/G) = \overline{K}$.

Définition 3.8 *Soit A_K un automate fini et co-accessible.*

Si $K = L(A_K)$, alors A_K représente K.

Si $K = L(A_K) \cap L_b(G)$ et $\overline{K} = \overline{L(A_K)} \cap L(G)$, alors A_K implante V.

Définition 3.9 *Soit $K, L \subseteq \Sigma^*$. Alors K et L sont dits non conflictuels si et seulement si*

$$\overline{K} \cap \overline{L} = \overline{K \cap L}.$$

Proposition 3.3 *Soit $E \subseteq \Sigma^*$ un langage $L_b(G)$-fermé.*

Soit $K = sup\mathcal{C}(E \cap L_b(G)) \neq \emptyset$.

Soit V un superviseur non bloquant tel que $L_b(V/G) = K$.

Soit A_K un automate accessible et co-accessible qui représente K.

Alors, A_K implante le superviseur V.

Proposition 3.4 *Soit A un automate co-accessible et G un générateur non bloquant tel que:*

1. *$L(A)$ est commandable par rapport à G, et*

2. *$L(A)$ et $L_b(G)$ sont non conflictuels.*

Alors, A implante un superviseur V non bloquant pour G tel que :

$$L_b(V/G) = L(A) \cap L_b(G).$$

Les proposition 3.3 et 3.4 nous permettent de faire la synthèse de superviseurs de deux façons différentes: soit en calculant un plus grand sous langage commandable

(proposition 3.3), soit en devinant la structure d'un superviseur (éventuellement plus simple que celui obtenu en suivant la proposition 3.3), et ensuite en vérifiant que les conditions de la proposition 3.4 sont remplies.

3.4.2 Observabilité

Considérons un système dynamique avec X comme ensemble d'états et $\alpha : X \longrightarrow X$ comme fonction de transition. Si $x_0 \in X$ est l'état initial du système, alors la trajectoire de ce dernier est de la forme:

$$x_0, x_1, ..., x_n, ...$$

où $x_{i+1} = \alpha(x_i), \forall i > 0$.

Soit $\gamma : X \longrightarrow Y$, une fonction de sortie. Alors, la séquence de sorties observée sera:

$$\gamma(x_0), \gamma(x_1), ...\gamma(x_n), ...$$

Définissons une relation d'équivalence ω sur X telle que: $x_0 \equiv x_0'(\text{ mod } \omega)$ pour deux états initiaux x_0, x_0' si et seulement si les séquences de sorties correspondant aux deux états sont les mêmes. En d'autres termes:

$$x_0 \equiv x_0'(\text{ mod } \omega) \Longleftrightarrow < \gamma(x_0), \gamma(\alpha(x_0)), ... > = < \gamma(x_0'), \gamma(\alpha(x_0')), ... > .$$

On dit que le système représenté par (X, α, γ) est observable si et seulement si $\omega = \perp$. C'est à dire si aucun état n'est équivalent à un autre. Dans ce cas ω est appelé l'observateur du système représenté par (X, α, γ).

3.5 Supervision modulaire

Soit $G = (\Sigma, Q, \delta, q_0, Q_b)$ un générateur et soit $E \subseteq L_b(G)$ un langage légal sur G. La synthèse des superviseurs telle que vue à la section 3.4 nous permet le calcul d'un plus grand sous langage commandable ($sup\mathcal{C}(E)$) en un temps polynomial en fonction du nombre d'états dans l'automate qui représente le langage légal $E \subseteq L_b(G)$ et du nombre d'états du générateur G. Cependant, la spécification formalisée par E est généralement déduite à partir de la conjoncture de plusieurs sous-spécifications et le SED modélisé par G est souvent composé de plusieurs sous-systèmes, et par conséquent le nombre d'états du système global croît de façcn exponentielle en fonction du nombre de composants (sous systèmes et sous spécifications). L'une des approches à ce problème est la synthèse modulaire des superviseurs. Cette approche est basée sur une décomposition du problème de synthèse en sous problèmes plus faciles à résoudre. L'idée fondamentale est de définir pour chaque sous système i un sous ensemble d'alphabet $\Sigma_i \subseteq \Sigma$ où Σ est l'ensemble d'alphabet du système global. Dans ce cas, Σ_i est appelé l'alphabet local du sous système i. Ainsi, la synthèse d'un superviseur est basée sur le fait que ce dernier ne contrôle que les évènement qui appartiennent à un Σ_i. Considérons un superviseur qui ne contrôle que les événements qui appartiennent à un alphabet local Σ_i. Ce superviseur est vu comme étant exécutant du contrôle d'un générateur local G_i tel que $L(G_i) = P_i(L(G))$, où P_i est la projection naturelle sur Σ_i (voir définition 3.4).

Formellement, un superviseur local est une application

$$V_i : P_i(L(G)) \longrightarrow \mathcal{P}(\Sigma_i),$$

telle que

$$V_i(t) \supseteq \Sigma_i \cap \Sigma_u, \forall t \in P_i(L(G)).$$

Puisqu'un superviseur V_i est indifférent à l'occurrence des événements de l'alphabet global mais qui n'appartiennent pas à Σ_i (c'est-à-dire les événements de $\Sigma \setminus \Sigma_i$), alors l'effet de V_i sur le système global G peut être donné par

$$\begin{aligned} \tilde{V}_i : \quad L(G) &\longrightarrow \mathcal{P}(\Sigma) \\ s &\mapsto V_i(P(s)) \cup (\Sigma \setminus \Sigma_i) \end{aligned}$$

Si le superviseur observe des événements qui n'appartiennent pas à l'alphabet local, alors il faut les inclure parmi ceux qui sont permis par le superviseur. Dans ce cas le superviseur est donné par $\tilde{V}_i : L(G) \longrightarrow \mathcal{P}(\Sigma) \cup \Sigma_o$, avec Σ_o est l'ensemble des événements de Σ observables, mais qui n'appartiennent pas à Σ_i.

L'effet de l'action concurrente de deux superviseurs locaux V_1 et V_2 est modélisé par la conjonction de $\tilde{V}_1$ et $\tilde{V}_2$, notée $\tilde{V}_1 \wedge \tilde{V}_2$ et définie comme suit:

$$\begin{aligned} \tilde{V}_1 \wedge \tilde{V}_2 : \quad L(G) &\longrightarrow \mathcal{P}(\Sigma) \\ s &\mapsto V_1(s) \cap V_2(s). \end{aligned}$$

Ce qui détermine l'efficacité de la synthèse modulaire des superviseurs est la *non conflictualité* des langages.

Définition 3.10 *Soit A_s un automate fini accessible et co-accessible. A_s est dit un superviseur propre si:*

1. *$L(A_s)$ est commandable, et*

2. *A_s/G est non bloquant:*

$$\overline{L_b(A_s/G)} = \overline{L(A_s) \cap L_b(G)} = \overline{L(A_s)} \cap L(G) = L(A_s/G).$$

Théorème 3.5 *Soit S_1 et S_2 deux superviseurs propres pour G. Alors,*

$$L_b((S_1 \bigwedge S_2)/G) = L_b(S_1/G) \cap L_b(S_2/G)$$

et de plus, $S_1 \bigwedge S_2$ est un superviseur propre pour G si et seulement si:

1. *$S_1 \bigwedge S_2$ est co-accessible, et*

2. *$L_b(S_1/G) et L_b(S_2/G)$ sont non conflictuels.*

Théorème 3.6 *Soit $K_1, K_2 \subseteq \Sigma^*$ commandables par rapport à G. Si K_1 et K_2 sont non conflictuels, alors $K_1 \cap K_2$ est commandable.*

Théorème 3.7 *Soit $E_1, E_2 \subseteq \Sigma^*$. Si $sup\mathcal{C}(E_1)$ et $sup\mathcal{C}(E_2)$ sont non conflictuels, alors*

$$sup\mathcal{C}(E_1 \cap E_2) = sup\mathcal{C}(E_1) \cap sup\mathcal{C}(E_1).$$

Tous ces résultats seront donc utiles pour pouvoir vérifier si la conjonction de plusieurs superviseurs (synthétisés de façon modulaire) produit un superviseur non bloquant par rapport au générateur G ou pas.

Chapitre 4

MODÉLISATION DU GÉNÉRATEUR ET SPÉCIFICATION DE QUELQUES EXEMPLES DE SERVICES TÉLÉPHONIQUES

Dans ce chapitre nous présentons quelques exemples de conflits dans les services téléphoniques. Nous avons choisi pour cela trois services téléphonique qui présentent des exemples bien connus du problème. Pour avoir des spécifications aussi réalistes que possible nous avons décidé de suivre la description de ces services dans la documentation fournie pour un concours qui a eu lieu dans le cadre du cinquième atelier international sur les intéractions de services (5th IWFI). Dans [GBGO98] nous trouvons l'ensemble de tous les exemples du concours.

Les exemples de générateurs pour la modélisation des systèmes téléphoniques qui ont été présentés dans [WTHM95] n'étaient pas conforme à la documentation du concours. Ainsi, il nous a fallu tout d'abord développer un nouveau générateur qui soit le plus simple possible mais qui reflète en même temps les réseaux téléphoniques réels.

Notons que les spécifications des services ont été développées de façon modulaire. En d'autres termes, nous avons réalisé des spécifications "naïves" qui ne tiennent pas compte des autres services qu'on pourrait vouloir implanter en même temps. Au contraire, nous cherchons à détecter les conflits par après.

Tous les automates du présent chapitre et tous les résultats ont été réalisés à l'aide du logiciel TCT [Won88], développé à l'Université de Toronto au sein de l'équipe Systems Control Group du département de génie électrique et de génie informatique. Cet outil fournit plusieurs fonctions utiles. Nous citons en particulier:

- La création d'un nouvel automate et l'édition d'un automate existant (en utilisant les commandes CREATE et EDIT).
- Le calcul d'un plus grand sous-langage commandable (en utilisant la commande SUPCON).
- Le calcul du produit synchrone de deux automates (en utilisant la commande SYNC).
- Le calcul d'un automate minimal équivalent à un automate donné (en utilisant la commande MINSTATE).

4.1 Adaptation du générateur

Le réseau téléphonique que nous considérons est formé de trois abonnés A,B et C. Le dévéloppement du générateur comprend plusieurs étapes:

1. la détermination des événements générés, ainsi que de leur observabilité et leur commandabilité par rapport à un abonné local (l'abonné A dans notre cas).

2. le dévéloppement d'un modèle local de l'abonné A et la déduction des deux autres modèles locaux.

3. le calcul du générateur local pour l'abonné A.

4.1.1 Événements générés par le réseau téléphonique

Le développement de systèmes complexes en ingénierie exige typiquement la décomposition de problèmes de désign en plusieurs aspects différents. Dans ce mémoire nous nous concentrons sur ces aspects du traitement des appels téléphoniques qui se prêtent à une analyse par une théorie de commande. Plus particulièrement, les événements auxquels nous nous somme intéressés sont ceux qui permettent de faire un changement dans l'état du réseau, en particulier, entre les liens entre les abonnés. Nous avons fait abstraction de tous les événements qui sont reliés à la signalisation: à savoir les différents types de sonneries, de tonalité et de "bips". Étant donné que nous ne nous intéressons pas au problème de facturation, nous avons omis tous les événements qui ne servent que pour établir les durées des communications. En plus, nous avons fait abstraction des événements qui ont pour rôle de renvoyer l'appel d'un abonné à un autre. Notre philosophie est que cet événement peut se traduire en une séquence des autres (si nous ne tenons pas compte de la facturation bien sûr). Les

événements engendrés par le générateur du réseau téléphonique sont:

1. $offhook_x$: décrochement du combiné de l'abonné x.

2. $flash_x$: un raccrochement suivi "rapidement" d'un décrochement du combiné de l'aboné x.

3. $dial_{xy}, (x \neq y)$: la demande de la part de x de l'établissement d'une communication avec y. Dans ce cas, x est l'appelant et y est le destinataire désiré. Cet événement regroupe en fait un ensemble d'événements élémentaires : la composition du numéro de y, et éventuellement, le déclenchement de la sonnerie (ou du bip) au niveau de y et le déclenchement la tonalité de demande au niveau de x.

4. $con_{xy}, (x \neq y)$: l'établissement de la communication proprement dite. Dans ce cas, les combinés de x et y sont forcément décrochés et il existe un lien physique actif entre les deux. Notons que dans ce cas, les deux abonnés peuvent être en communication, mais l'un d'entre eux est mis en "attente" par l'autre.

5. $nocon_{xy}, (x \neq y)$: signalisation de l'impossibilité d'acheminer un appel de x vers y dû soit à un problème dans le réseau lui-même, soit à l'occupation de la ligne du destinataire. Un con_{xy} peut résulter d'un $dial_{xz}$ avec $z \neq y$ (dans le cas d'un transfert d'appel).

6. con_{xyz} : $x, y, z \in \{A, B, C\}$ avec x, y et z sont différents: l'établissement de la communication entre x et z suite à un $dial_{xy}$ (cas d'un transfert d'appel).

Nous présentons dans les tableaux 4.1, 4.2 et 4.3 l'ensemble des événements locaux ou observables (noté $\Sigma_i, i \in \{A, B, C\}$) aux niveaux de chaque commutateur i ainsi que leurs commandabilités par rapport au commutateur A.

Num	Événement	Commandabilité	Code TCT
1	offhook A	N	102
2	flash A	N	104
3	onhook A	N	106
4	onhook B	N	206
5	onhook C	N	306
6	dial AB	O	121
7	dial AC	O	131
8	dial BA	N	212
9	dial CA	N	312
10	con AB	N	124
11	con AC	N	134
12	con BA	O	213
13	con CA	O	313
14	nocon AB	N	126
15	nocon AC	N	136
16	nocon BA	O	215
17	nocon CA	O	315
18	con BAC	O	237
19	con CAB	O	427

Tableau 4.1: Liste des événements au commutateur A.

4.1.2 Le modèle du commutateur A

Nous considérons que le modèle de A permute entre deux états majeurs. Un état de raccrochement (état 0) et un état de décrochement (état 1). L'état 0 est l'état initial ainsi que le seul état but (ou marqué). À l'état 1, l'automate reste inchangé lors de l'événement $flash_A$. Notons qu'avec cette représentation, nous offrons une solution triviale du problème de blocage qui peut se produire, étant donné que l'abonné peut toujours éviter le blocage en faisant un *onhook* pour retourner à son état but. Nous voulons que le modèle soit tel que l'invariant suivant est toujours vérifié dans tout

Num	Événement	Commandabilité	Code TCT
1	offhook B	N	202
2	flash B	N	204
3	onhook A	N	106
4	onhook B	N	206
5	onhook C	N	306
6	dial AB	O	121
7	dial BC	N	232
8	dial BA	N	212
9	dial CB	N	322
10	con AB	N	124
11	con BC	N	234
12	con BA	O	213
13	con CB	N	324
14	nocon AB	N	126
15	nocon BC	N	236
16	nocon BA	O	215
17	nocon CB	O	326
18	con CBA	N	618
19	con ABC	N	138

Tableau 4.2: Liste des événements au commutateur B.

mot s du langage achevé:

$$n(dial_{AB}) + n(dial_{AC}) = n(con_{AB}) + n(con_{AC}) + n(nocon_{AB}) + n(nocon_{AC}).$$

où $n(\sigma)$ désigne le nombre d'occurrences de l'événement σ dans le mot s. Un autre invariant trivial de notre modèle est

$$n(onhook_A) = n(offhook_A).$$

Pour cela l'ajout d'un autre état s'avère nécessaire.

Le nouvel état (2) permettra donc d'assurer le premier invariant. Le modèle de

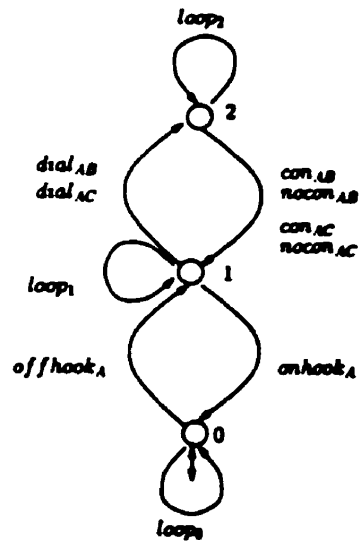
Num	Événement	Commandabilité	Code TCT
1	offhook C	N	302
2	flash C	N	304
3	onhook A	N	106
4	onhook B	N	206
5	onhook C	N	306
6	dial CB	N	322
7	dial AC	O	131
8	dial BC	N	232
9	dial CA	N	312
10	con CB	N	324
11	con AC	N	134
12	con BC	N	234
13	con CA	O	313
14	nocon CB	N	326
15	nocon AC	N	136
16	nocon BC	N	236
17	nocon CA	O	315
18	con ACB	N	328
19	con BCA	N	518

Tableau 4.3: Liste des événements au commutateur C.

l'abonné A, appelé $\mathcal{SWITCH}_A$, est donné dans la figure 4.1. $\mathcal{SWITCH}_A$ est un automate de 3 états et 18 transitions.

Le passage de l'état 1 à l'état 2 se fait lors de l'occurrence des événements $dial_{AB}$ ou $dial_{AC}$.

Le retour de l'état 2 à l'état 1 se fait lors d'un des événements con_{AB}, con_{AC}, $nocon_{AB}$ ou $nocon_{AC}$. En plus, et pour avoir un modèle plus réaliste, nous supposons que la génération des événements con_{BA} et con_{CA} ne peut se produire que si le combiné de A est décroché, c'est à dire lorsque le générateur est à l'état 1. En plus d'être relativement simple, le modèle que nous proposons représente implicitement le renvoi d'appel par la génération d'un événement con_{AB} (resp. con_{AC}) à la suite de l'occurrence de



$loop_0 = \{con_{iAj} : i, j \in \{B,C\}\}$

$loop_1 = \{flash_A, con_{iAj}, con_{Aij}, con_{iA} : i, j \in \{B,C\}\}$

$loop_2 = \{flash_A, con_{iAj} : i, j \in \{B,C\}\}$

Figure 4.1: Modèle du générateur local A

l'événement $dial_{AC}$ (resp. $dial_{AB}$). L'occurrence de l'événement $flash_A$ à l'un des états 1 ou 2 laisse l'automate dans son état. En plus, l'occurrence des événements con_{BAC} et con_{CAB} à n'importe quel état de l'automate laisse ce dernier dans son état. Nous obtenons ainsi:

$$loop_0 = \{con_{iAj} : i, j \in \{B,C\}, i \neq j\}$$

$$loop_1 = \{flash_A.con_{iA}.con_{Aij}, con_{iAj} : i, j \in \{B,C\}, i \neq j\}$$

et

$$loop_2 = \{flash_A.con_{iAj} : i, j \in \{B,C\}, i \neq j\}.$$

4.1.3 Calcul du générateur local

Dans une première étape, nous formalisons les deux autres modèles locaux des commutateurs B et C appelés $\mathcal{SWITCH}_B$ et $\mathcal{SWITCH}_C$ respectivement. Chacun de ces automates est formé de 3 états et 18 transitions. Ensuite et avec la commande **SYNC** de TCT, nous calculons le générateur global des trois abonnés. En fait, le générateur global est le produit synchrone des trois modèles locaux. Nous obtenons un automate de 27 états et 385 transitions appelé $\mathcal{GLOBAL}$. Pour déterminer le générateur local du commutateur A, nous utilisons la commande **PROJECT** avec le générateur du réseau global comme automate et la liste des événements non observables par le commutateur A comme liste d'événements à "effacer":

$$LIST = \{offhook_i, flash_i, dial_{ij}, con_{ij}, con_{Aij}, con_{ijA}, nocon_{ij} : i, j \in \{B, C\}\}, i \neq j.$$

L'automate obtenu, appelé $\mathcal{LOCAL}_A$, contient 15 états et 180 transitions.

4.2 Amélioration du générateur

Après avoir dévéloppé la première version de notre générateur du réseau téléphonique, nous avons pensé qu'il serait utile de l'améliorer de sorte à avoir un modèle plus réaliste. Le problème est que dans les réseaux téléphoniques, un abonné A qui est en communication avec un autre ne peut initialiser un autre appel qu'après l'occurrence de l'événement $flash_A$ (ou $onhook_A$ suivie d'un $offhook_A$). Le générateur que nous avons présenté dans la section précédente ne respecte pas cette contrainte. Pour cela nous jugeons qu'il est important de modéliser une spécification qui respecte cette contrainte. Pour cette raison, nous présentons un automate de deux états (numérotés

0 et 1) et 29 transitions que nous avons dévéloppé, appelé $\mathcal{FLASH}_A$. (voir figure 4.2)

- L'état 0 est l'état initial, et aussi le seul état marqué. Nous passons à l'état 1 s'il y a occurrence d'un des événements suivants:

$$\{con_{Ai}, nocon_{Ai}, con_{iA} : i \in \{B, C\}\}.$$

Donc cet état (l'état 0) présente la situation où l'abonné A n'est pas en communication avec un autre abonné. Ainsi, l'automate $\mathcal{FLASH}_A$ reste dans cet état à l'occurrence de des événements de la liste suivante:

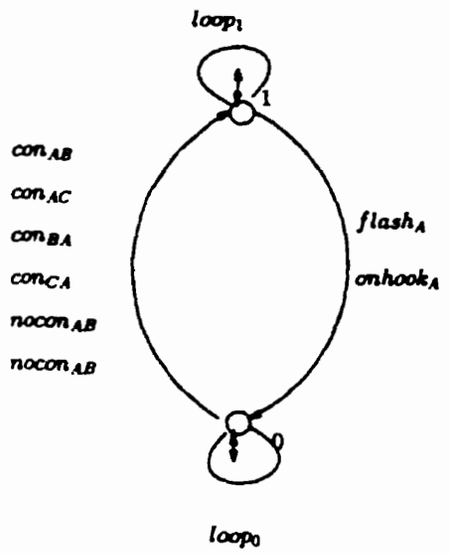
$$loop_0 = \Sigma_A \setminus \{con_{Ai}, nocon_{Ai}, con_{iA} : i \in \{B, C\}\}.$$

- L'état 1 correspond au cas où l'abonné A est en communication avec (au moins) un abonné. L'occurrence d'un des événements $flash_A$ ou $onhook_A$ amène l'automate à son état 0, alors que les événements suivants le laissent dans son état 1

$$loop_1 = \{onhook_i, dial_{iA}, nocon_{iA}, con_{iAj} : i, j \in \{B, C\}, i \neq j\}.$$

La spécification que nous avons dévéloppée possède les propriétés suivantes:

1. $\mathcal{FLASH}_A$ est co-accessible.
2. $L(\mathcal{FLASH}_A)$ est commandable par rapport à $L(\mathcal{LOCAL}_A)$.
3. $L(\mathcal{FLASH}_A)$ et $L_b(\mathcal{LOCAL}_A)$ sont non conflictuels.



$$loop_0 = \Sigma_A \setminus \{con_{Ai}, nocon_{Ai}, con_{iA} : i \in \{B, C\}\}$$

$$loop_1 = \{con_{BAC}, con_{CAB}, onhook_i, dial_{iA}, nocon_{iA} : i \in \{B, C\}\}$$

Figure 4.2: Amélioration du générateur local de A

La façon de vérifier de telles propriétés avec TCT sera présentée ultérieurement dans la section 4.6. Nous pouvons d'après la proposition 3.4 déduire que l'automate $\mathcal{FLASH}_A$ implante un superviseur non bloquant V_{FLASH} pour $\mathcal{LOCAL}_A$ tel que:

$$L_b(V_{FLASH}/\mathcal{LOCAL}_A) = L(\mathcal{FLASH}_A) \cap L_b(\mathcal{LOCAL}_A).$$

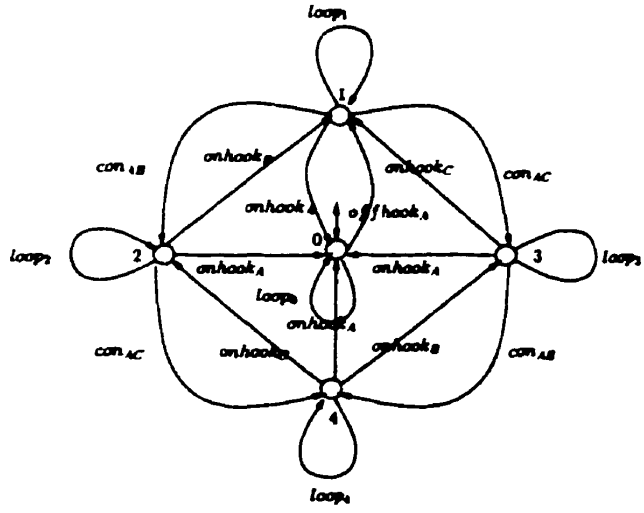
Nous considérons dans tout ce qui suit que le générateur $\mathcal{GEN}_A$ désigne le générateur local de A sous la supervision de V_{FLASH}. $\mathcal{GEN}_A$ est un automate de 20 états et 230 transitions.

Nous présentons maintenant les spécifications que nous avons développées pour les trois services téléphoniques: "*conférence à trois*", noté TWC (Three Way Calling), "*appel en attente*", noté CW (Call Waiting) et "*transfert d'appel, ligne occupée*", noté CFBL (Call Forwarding Busy Line). L'analyse de ces spécifications et de leurs propriétés sera entreprise ultérieurement dans ce chapitre.

4.3 Spécification du service "conférence à trois"

Ce service permet à un abonné A d'être en communication avec deux autres abonnés en même temps. Dans le cas de ce service le même abonné doit initialiser les deux appels. Nous supposons dans le cadre de notre spécification que A est l'abonné qui initialise les appels avec les abonnés B et C.

Notre spécification comprend essentiellement deux "branches": une branche qui modélise une communication avec B suivie d'une communication avec C, et une deuxième branche qui modélise une communication avec C suivie d'une communication avec B. L'automate que nous présentons, appelé $\mathcal{SPEC}_{TWC}$, contient cinq états numérotés de



$$loop_0 = \{flash_A, onhook_i, dial_{iA}, nocon_{iA} : i \in \{B, C\}\}$$

$$loop_1 = loop_2 = loop_3 = \{flash_A, dial_{Ai}, dial_{iA}, nocon_{Ai}, nocon_{iA}, onhook_i : i \in \{B, C\}\}$$

$$loop_4 = \{flash_A, dial_{iA}, nocon_{iA} : i \in \{B, C\}\}$$

Figure 4.3: Spécification du service TWC

0 à 4 (voir figure 4.3).

L'état initial (état 0) est celui pour lequel le combiné de l'abonné A est raccroché. De n'importe quel état de l'automate, l'occurrence de l'événement $onhook_A$ amène l'automate à son état initial, qui est en plus son seul état marqué. L'automate reste dans l'état 0 en l'occurrence des événements suivants:

$$loop_0 = \{flash_A, onhook_i, dial_{iA}, nocon_{iA} : i \in \{B, C\}\}.$$

Lorsque l'abonné A décroche son combiné, l'automate passe à l'état 1. De cet état, nous pouvons passer ou bien à l'état 2, lorsque le premier appel est effectué avec l'abonné B (c'est-à-dire lors de l'événement con_{AB}) ou bien à l'état 3 lorsque le

premier appel est effectué avec l'abonné C (c'est-à-dire lors de l'événement con_{AC}). Les événements suivants laissent l'automate dans son état 1:

$$loop_1 = \{flash_A, dial_{Ai}, dial_{iA}, nocon_{Ai}, nocon_{iA}, onhook_i : i \in \{B, C\}\}.$$

Une fois rendu à l'état 2 (resp. 3), si l'abonné B (resp. C) raccroche, le système retourne à l'état 1, à partir duquel l'abonné A peut soit initialiser un autre appel, soit tout simplement raccrocher. À partir de l'état 2 (resp. 3), l'occurrence de l'événement con_{AC} (resp. con_{AB}) amène l'automate à l'état 4. Dans ce dernier état, les trois abonnés sont en communication. Les événements qui gardent l'automate dans son état 2 (resp. 3) sont:

$$loop_2 = loop_3 = loop_1 =$$

$$\{flash_A, dial_{Ai}, dial_{iA}, nocon_{Ai}, nocon_{iA}, onhook_i : i \in \{B, C\}\}.$$

L'état 4 est l'état dans lequel les trois abonnés sont en communication, c'est-à dire en "conférence". À partir de cet état les seuls événements qui sont autorisés sont $onhook_B$ qui ramène l'automate à l'état 3, et $onhook_C$ qui ramène l'automate à l'état 2 à partir desquels l'abonné A peut initialiser une autre "conférence". Comme d'habitude l'occurrence de l'événement $onhook_A$ amène l'automate à son état initial. Les seuls autres événements qui sont autorisés et qui gardent l'automate dans son état 4 sont:

$$loop_4 = \{flash_A, dial_{iA}, nocon_{iA} : i \in \{B, C\}\}.$$

En résumé:

- l'état 0 correspond au cas où le combiné de l'abonné A est raccroché.
- l'état 1 correspond au cas où le combiné de l'abonné A est décroché sans qu'il soit en communication avec un autre abonné.
- l'état 2 correspond au cas où A est en communication avec B.
- l'état 3 correspond au cas où A est en communication avec C.
- l'état 4 correspond au cas où A est en conversation avec B et C en même temps.

Soulignons que cette spécification ne constitue qu'une contrainte sur le comportement du service; elle ne spécifie pas comment implanter le service. Vu que nous ne nous intéressons qu'aux aspects "commande" du service, nous avons tenu compte uniquement aux états de A (raccroché ou décroché) et des liens entre l'abonné A et les deux autres abonnés (en communication ou pas).

4.4 Spécification du service "appel en attente"

Ce service permet à un abonné A en communication avec un autre de mettre ce dernier en "attente" pour recevoir un appel d'un autre abonné. Dans le cas de ce service, l'abonné A peut être soit demandeur, soit demandé dans le premier appel. Mais il ne peut être que demandé dans le deuxième appel. Comme d'habitude, nous réalisons la spécification au niveau du commutateur de A. La spécification tient compte des deux cas suivants:

1. A est en communication avec B, puis il met B en attente pour recevoir un appel de C.

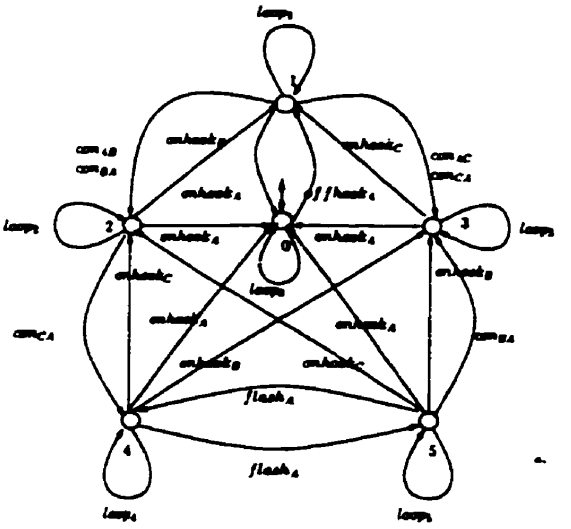


Figure 4.4: Spécification du service CW

2. A est en communication avec C, puis il met C en attente pour recevoir un appel de B.

L'automate que nous proposons, appelé $\mathcal{SPEC}_{CW}$, contient 6 états numérotés $0, ..., 5$ (voir figure 4.4).

L'état initial (état 0) est celui pour lequel le combiné de l'abonné A est raccroché. De n'importe quel état de l'automate, l'occurrence de l'événement $onhook_A$ amène l'automate à son état initial, qui est en plus son seul état marqué. L'automate reste dans l'état en l'occurrence de la liste des événements suivants:

$$loop_0 = \{flash_A, onhook_i, dial_{iA}, nocon_{iA} : i \in \{B, C\}\}.$$

Lorsque l'abonné A décroche son combiné, l'automate passe à l'état 1. De cet état, nous pouvons passer ou bien à l'état 2, lorsque le premier appel est effectué avec l'abonné B, c'est-à-dire l'occurrence de l'événement con_{AB} ou l'événement con_{BA}, ou bien à l'état 3 lorsque le premier appel est effectué avec l'abonné C, c'est-à-dire l'occurrence de l'événement con_{AC} ou l'événement con_{CA}. La liste des événements suivants laissent l'automate dans son état 1:

$$loop_1 = \{flash_A, dial_{Ai}, dial_{iA}, nocon_{Ai}, nocon_{iA}, onhook_i : i \in \{B, C\}\}.$$

Une fois rendu à l'état 2 (resp. 3), si l'abonné B (resp. C) raccroche, le système retourne à l'état 1, à partir duquel l'abonné A peut soit initialiser un autre appel, soit tout simplement raccrocher. À partir de l'état 2 (resp. 3), l'occurrence de l'événement con_{CA} (resp. con_{BA}) amène l'automate à l'état 4 (resp. l'état 5). Les événements qui gardent l'automate dans son état 2 (resp. 3) sont:

$$loop_2 = loop_3 = loop_1 =$$

$$\{flash_A, dial_{Ai}, dial_{iA}, nocon_{Ai}, nocon_{iA}, onhook_i : i \in \{B, C\}\}.$$

L'état 4 correspond au cas où l'abonné B est en attente et A est en conversation avec C. L'état 5 correspond au cas où l'abonné C est en attente et A est en conversation avec B. L'abonné A peut basculer entre ces deux états avec des $flash_A$. À partir de l'un des états 4 ou 5, l'occurrence de l'événement $onhook_B$ amène l'automate à l'état 3 et l'occurence de l'événement $onhook_C$ amène l'automate à l'état 2. Les événements

qui gardent l'automate dans son état actuel (4 ou 5) sont:

$$loop_4 = loop_5 = \{dial_{iA}, nocon_{iA} : i \in \{B, C\}\}.$$

En résumé:

- l'état 0 correspond au cas où le combiné de l'abonné A est raccroché.
- l'état 1 correspond au cas où le combiné de l'abonné A est décroché sans qu'il soit en communication avec un autre abonné.
- l'état 2 correspond au cas où A est en communication avec B.
- l'état 3 correspond au cas où A est en communication avec C.
- l'état 4 correspond au cas où A est en communication avec C, mais B est en attente.
- l'état 5 correspond au cas où A est en communication avec B, mais C est en attente.

Comme dans le cas du service TWC, nous nous intéressons uniquement aux états de A (décroché ou raccroché) et des liens entre A et les abonnés pour la réalisation de l'automate (en communication, pas de communication, en attente).

4.5 Spécification du service "transfert d'appel, ligne occupée"

Étant donné que con_{ij} pour $i, j \in \{B, C\}$ ne sont pas des événements de l'alphabet local de A, nous avons enrichi notre modèle par les deux événements locaux et com-

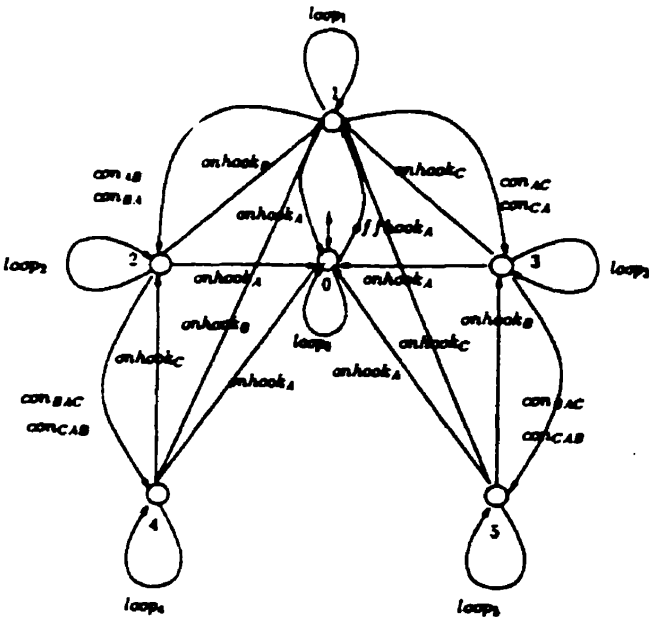


Figure 4.5: Spécification du service CFBL

mandables (par rapport à A) con_{BAC} et con_{CAB} qui traduisent le renvoi d'un appel destiné à A vers un autre abonné. (voir tableau 4.1). Ces deux événements traduisent respectivement les séquences d'événements suivantes $dial_{BA}.con_{BC}$ et $dial_{CA}.con_{CB}$. L'automate que nous proposons pour la spécification du service contient 6 états numérotés $0, ..., 5$ (voir figure 4.5). L'état initial (état 0) est celui pour lequel le combiné de l'abonné A est raccroché. De n'importe quel autre état de l'automate, l'occurrence de l'événement $onhook_A$ amène l'automate à son état initial, qui est en plus son seul état marqué. L'automate reste dans l'état 0 lors des événements suivants:

$$loop_0 = \{flash_A.onhook_i, dial_{iA}, nocon_{iA} : i \in \{B,C\}\}.$$

Lorsque l'abonné A décroche son combiné, l'automate passe à l'état 1. De cet état, nous pouvons passer ou bien à l'état 2, lorsque l'appel est effectué avec l'abonné B (c'est-à-dire lors de l'événement con_{AB} ou de l'événement con_{BA}), ou bien à l'état 3 lorsque l'appel est effectué avec l'abonné C (c'est-à-dire lors de l'événement con_{AC} ou de l'événement con_{CA}). Les événements suivants laissent l'automate dans son état 1:

$$loop_1 = \{flash_A, dial_{iA}, nocon_{Ai}, nocon_{iA}, onhook_i : i \in \{B, C\}\}.$$

Une fois rendu à l'état 2(resp. 3), si l'abonné B (resp. C) raccroche, le système retourne à l'état 1, à partir duquel l'abonné A peut soit initialiser un autre appel, soit tout simplement raccrocher. À partir de l'état 2 (resp. 3), l'occurrence de l'événement con_{BAC} ou con_{CAB}, suite à une nouvelle demande de communication avec A, amène l'automate à l'état 4 (resp. l'état 5). Les événements qui gardent l'automate dans son état 2 (resp. 3) sont:

$$loop_2 = loop_3 =$$

$$\{flash_A, dial_{Ai}, dial_{iA}, nocon_{iA}, onhook_i : i \in \{B, C\}\}.$$

L'état 4 (resp. 5) correspond au cas où l'abonné B est en conversation avec C après une demande d'appel avec A ($dial_{BA}$ ou $dial_{CA}$). À partir de l'état 4 (resp. 5), l'occurrence de l'événement $onhook_B$ (resp. $onhook_C$ amène l'automate à l'état 1 et l'occurrence de l'événement $onhook_C$ (resp. $onhook_B$) amène l'automate à l'état 2 (resp. 3). Les événements qui gardent l'automate dans son état actuel (4 ou 5) sont:

$$loop_4 = loop_5 = \{flash_A, dial_{iA}, nocon_{iA} : i \in \{B, C\}\}.$$

4.6 Propriétés des spécifications dévéloppées

Présentons tout d'abord l'ensemble des automates que nous avons dévéloppés.

1. $\mathcal{GEN}_A$ est le générateur local pour l'abonné A (tel que défini à la section 4.2).
2. $\mathcal{SPEC}_{TWC}$ est l'automate dévéloppé pour la spécification du service "conférence à trois". Il est formé de 5 états et 58 transitions.
3. $\mathcal{SPEC}_{CW}$ est l'automate dévéloppé pour la spécification du service appel en attente. Il est formé de 6 états et 58 transitions.
4. $\mathcal{SPEC}_{CFBL}$ est l'automate dévéloppé pour la spécification du service "transfert d'appel, ligne occupée". Il est formé de 6 états et 64 transitions.

Nous cherchons à voir si les conditions de la proposition 3.4 sont remplies par les spécifications que nous avons dévéloppées.
Soit $SPEC \in \{\mathcal{SPEC}_{TWC}, \mathcal{SPEC}_{CW}, \mathcal{SPEC}_{CFBL}\}$. Nous allons vérifier si l'automate $SPEC$ vérifie les conditions suivantes:

1. $SPEC$ est co-accessible.
2. $L(SPEC)$ est commandable par rapport au langage engendré $L(\mathcal{GEN}_A)$.
3. $L(SPEC)$ et $L_b(\mathcal{GEN}_A)$ sont non conflictuels.

Si ces conditions sont remplies nous pouvons alors déduire que $SPEC$ implante un superviseur non bloquant pour $\mathcal{GEN}_A$.

La commande **TRIM(G)** de TCT calcule la composante accessible et co-accessible de l'automate G. Soient $TTWC$, TCW, $TCFBL1$ et $TCFBL2$ les composantes accessibles et co-accessibles des automates $\mathcal{SPEC}_{TWC}$, $\mathcal{SPEC}_{CW}$ et $\mathcal{SPEC}_{CFBL}$ respectivement. Nous pouvons maintenant facilement vérifier avec la commande **ISOMORPH**

de TCT que les automates développés sont accessibles et co-accessibles étant donné qu'ils sont identiques à leurs composantes accessibles et co-accessibles.

Vérifions maintenant la deuxième condition. La commande **CONDAT(G,A)** de TCT permet de déterminer l'ensemble des événements générés par G mais qui sont empêchés par A dans chaque état.

Étant donné que les spécifications ont été développées de manière "naïves", les deux événements con_{iAj} $(i, j \in \{B, C\}, i \neq j)$ sont interdits dans tous les états des automates $\mathcal{SPEC}_{TWC}$ et $\mathcal{SPEC}_{CW}$. Pour l'automate $\mathcal{SPEC}_{TWC}$, aucun autre événement n'est interdit à l'état 0. Les événements qui sont interdits dans les autres états sont les con_{iA}, avec $i \in \{B, C\}$. En plus, il y a deux autres événements qui sont interdits à l'état 4 qui sont $dial_{Ai}$, avec $i \in \{B, C\}$. Ceci est justifiable étant donné qu'à cet état l'abonné A est déjà en communication avec les deux autres abonnés. Nous remarquons que les seuls événements interdits dans $\mathcal{SPEC}_{TWC}$ sont des événements commandables et par conséquent, $L(\mathcal{SPEC}_{TWC})$ est commandable par rapport à $L(\mathcal{GEN}_A)$.

Pour l'automate $\mathcal{SPEC}_{CW}$, les seuls événements interdits aux états 0 et 1 sont les con_{iAj}. Les événements $dial_{Ai}$, avec $i \in \{B, C\}$ sont interdits dans tous les autres états. À l'état 2 nous avons – en plus – interdit l'événement con_{BA}, étant donné qu'à cet état l'abonné A est déjà en communication avec l'abonné B. Il est en de même à l'état 3 où nous avons interdit l'événement con_{CA}. Etant donné qu'aux états 4 et 5 les trois abonnés sont en communication, nous avons interdit en plus les événements con_{iA}, avec $i \in \{B, C\}$. Encore une fois, le langage $L(\mathcal{SPEC}_{CW})$ est commandable par rapport à $L(\mathcal{GEN}_A)$ étant donné que tous les événements interdits sont commandables.

Pour l'automate $\mathcal{SPEC}_{CFBL}$, les seuls événements interdits aux états 0 et 1 sont con_{BAC} et con_{CAB}. Ceci s'explique par le fait qu'à ces états l'abonné A n'est pas en communication avec un autre abonné. Les événements $dial_{Ai}$ et con_{iA} sont interdits dans tous les autres états. Informellement, ceci signifie que A ne peut ni initialiser un autre appel ni recevoir un autre appel une fois qu'il est en communication avec un autre abonné. Aux états 4 et 5, les trois abonnés sont en communication. Ainsi, nous avons interdit en plus, les événements con_{BAC} et con_{CAB}. Dans cet automate, tous les événements interdits sont des événements commandables, et par conséquent, $L(\mathcal{SPEC}_{CFBL})$ est commandable par rapport à $L(\mathcal{GEN}_A)$. En conclusion, les trois automates développés pour les services téléphoniques étudiés reconnaissent des langages commandables par rapport à $L(\mathcal{GEN}_A)$.

La commande **NONCONFLICT**(A_1,A_2) vérifie si les deux langages achevés de A_1 et A_2 ($L_b(A_1)$ et $L_b(A_2)$) sont conflictuels ou pas.
L'exécution des commandes:
- **NONCONFLICT**($\mathcal{GEN}_A$,$\mathcal{SPEC}_{TWC}$),
- **NONCONFLICT**($\mathcal{GEN}_A$,$\mathcal{SPEC}_{CW}$) et
- **NONCONFLICT**($\mathcal{GEN}_A$,$\mathcal{SPEC}_{CFBL}$)

nous montrent que chacun des langages des spécifications n'est pas en conflit avec le langage marqué du générateur local $L_b(\mathcal{GEN}_A)$.

Toutes les conditions de la proposition 3.4 sont remplies. Ainsi, nous pouvons conclure que chacun des automates développés $\mathcal{SPEC}_{TWC}$, $\mathcal{SPEC}_{CW}$ et $\mathcal{SPEC}_{CFBL}$ implante un superviseur non bloquant pour $\mathcal{GEN}_A$. Soient V_{TWC}, V_{CW} et V_{CFBL2} des superviseurs implantés par les automates $\mathcal{SPEC}_{TWC}$, $\mathcal{SPEC}_{CW}$ et $\mathcal{SPEC}_{CFBL}$ respec-

tivement. Alors:

$$L_b(V_{TWC}/\mathcal{GEN}_A) = L(\mathcal{SPEC}_{TWC}) \cap L_b(\mathcal{GEN}_A)$$

$$L_b(V_{CW}/\mathcal{GEN}_A) = L(\mathcal{SPEC}_{CW}) \cap L_b(\mathcal{GEN}_A)$$

$$L_b(V_{CFBL2}/\mathcal{GEN}_A) = L(\mathcal{SPEC}_{CFBL}) \cap L_b(\mathcal{GEN}_A).$$

Avec TCT, l'intersection de deux automates A_1 et A_2 se calcule à l'aide de la commande **MEET**(A_1, A_2).

Dans [WTHM95], une approche basée sur des langages achevés multiples a été introduite pour vérifier la validité de la modélisation. Dans cette approche, plusieurs langages achevés co-existent et chaque langage achevé est interprété comme une nouvelle spécification qui s'ajoute à la spécification initiale. Il faut qu'il existe un superviseur non bloquant pour chacun de ces langage achevés. Voyons à quoi correspond ceci dans nos spécifications.

1. Dans l'automate $\mathcal{SPEC}_{TWC}$, l'état 4 correspond au cas où les trois abonnés sont en "conférence", et c'est le seul état qui représente cette situation. Ainsi, nous marquons cet état et nous obtenons un nouvel automate que nous appelons $STWC$.

2. Dans l'automate $\mathcal{SPEC}_{CW}$, les états 4 et 5 correspondent tous les deux au cas où l'abonné A est en conversation avec un autre abonné alors que le troisième est mis en attente. Donc ces deux états sont les seuls états qu'il faut marquer. Nous appelons l'automate obtenu SCW.

3. Dans l'automate $\mathcal{SPEC}_{CFBL}$, les états 4 et 5 correspondent au cas où l'appel vers A est transféré à un autre abonné. Donc ces deux états sont les seuls états

à marquer. Nous obtiendrons ainsi un nouvel automate $SCFBL$.

Maintenant, nous marquons tous les états du générateur local. Nous obtenons un nouveau générateur GEN_AM, tel que:

$$L_b(GEN_AM) = \overline{L_b(\mathcal{GEN}_\mathcal{A})}.$$

En effectuant les tests de non conflictualité avec TCT, nous obtenons les résultats suivant:

$$\mathbf{NONCONFLICT(GEN_AM, STWC)} = TRUE.$$

$$\mathbf{NONCONFLICT(GEN_AM, SCW)} = TRUE.$$

$$\mathbf{NONCONFLICT(GEN_AM, SCFBL)} = TRUE.$$

Informellement, ces résultats confirment que l'état 4 de $\mathcal{SPEC}_{TWC}$, les états 4 et 5 de $\mathcal{SPEC}_{CW}$, et les états 4 et 5 de $\mathcal{SPEC}_{CFBL}$ sont toujours atteignables et que le système n'est jamais bloqué à cause des langages achevés additionnels.

Chapitre 5

TESTS D'INTERFÉRENCES ENTRE LES SERVICES TÉLÉPHONIQUES ET DISCUSSION DES RÉSULTATS

Dans une première partie de ce chapitre, nous allons utiliser le générateur et les spécifications dévéloppées dans le chapitre précédent pour faire des tests d'interférences entre les trois services TWC, CW et CFBL. Dans une deuxième partie, nous allons analyser et discuter les résultats obtenus. Les résultats ont été obtenus à l'aide de l'outil logiciel TCT [Won88].

5.1 Tests d'interférences entre les trois services téléphoniques

La présence de plusieurs services dans un commutateur téléphonique peut poser des problèmes de compatibilité entre ces différents services en raison de leurs "interférences". La théorie des systèmes à événements discrets nous donne un outil de détection de ces interférences basé sur la notion de langages conflictuels.

Nous disons que deux langages $K_1, K_2 \subseteq \Sigma^*$ sont non interférents si $\overline{K_i}$ et K_j sont non conflictuels, avec $(i, j \in \{1,2\}, i \neq j)$.

Cette condition est plus faible que la condition de non conflictualité de deux langages. Soit A_i un automate qui reconnaît un langage K_i. Alors l'automate A_{i_m} obtenu à partir de A_i en marquant tous les états reconnaît le langage $\overline{K_i}$. Considérons donc les automates $STWC_m$, SCW_m et $SCFBL_m$ obtenus resp ectivement à partir de $\mathcal{SPEC}_{TWC}$, $\mathcal{SPEC}_{CW}$ et $\mathcal{SPEC}_{CFBC}$ en marquant tous les états.

5.1.1 Test d'interférence entre les services "conférence à trois" et "appel en attente"

Soit $TWCCW_m$ la conjonction des deux superviseurs $STWC_m$ et SCW_m. Cet automate peut être obtenu en exécutant la commande

$$\mathbf{TWCCW_m = MEET(STWC_m, SCW_m)}$$

de TCT. Puisque $\mathcal{SPEC}_{TWC}$ et $\mathcal{SPEC}_{CW}$ sont co-accessibles alors:

$$L_b(TWCCW_m) = \overline{L_b(\mathcal{SPEC}_{TWC})} \cap \overline{L_b(\mathcal{SPEC}_{CW})}.$$

Étant donné que $\mathcal{SPEC}_{TWC}$ et $\mathcal{SPEC}_{CW}$ sont non conflictuels

$$\mathbf{NONCONFLICT}(\mathcal{SPEC}_{TWC}, \mathcal{SPEC}_{CW}) = \mathbf{TRUE}$$

Alors nous pouvons déduire d'après la définition 3.9 que:

$$\overline{L_b(\mathcal{SPEC}_{TWC})} \cap \overline{L_b(\mathcal{SPEC}_{CW})} = \overline{L_b(\mathcal{SPEC}_{TWC}) \cap L_b(\mathcal{SPEC}_{CW})}$$

et par conséquent

$$L_b(TWCCW_m) = \overline{L_b(\mathcal{SPEC}_{TWC}) \cap L_b(\mathcal{SPEC}_{CW})}.$$

Soit $GTWC_m$ l'automate qui représente le fonctionnement en boucle fermée du générateur GEN_AM sous la supervision de $STWC_m$.

$$\mathbf{GTWC_m} = \mathbf{MEET}(\mathbf{STWC_m}, \mathbf{GEN_AM}).$$

Soit GCW_m l'automate qui représente le fonctionnement en boucle fermée du générateur GEN_AM sous la supervision de SCW_m.

$$\mathbf{GCW_m} = \mathbf{MEET}(\mathbf{SCW_m}, \mathbf{GEN_AM}).$$

Remarquons que puisque $L_b(\mathcal{SPEC}_{TWC})$ et $L_b(\mathcal{GEN}_A)$ sont non conflictuels, alors

$$\begin{aligned} L_b(GTWC_m) &= \overline{L_b(\mathcal{SPEC}_{TWC}/GEN_A)} \\ &= \overline{L_b(\mathcal{GEN}_A) \cap L_b(\mathcal{SPEC}_{TWC})} \\ &= \overline{L_b(\mathcal{GEN}_A)} \cap \overline{L_b(\mathcal{SPEC}_{TWC})}. \end{aligned}$$

Effectuons les tests de non conflictualité suivants pour voir si les deux services "conférence à trois" et "appel en attente" sont interférents selon les spécifications que nous avons dévéloppées.

1. $\overline{L_b(\mathcal{GEN}_A) \cap \overline{L_b(\mathcal{SPEC}_{TWC}) \cap L_b(\mathcal{SPEC}_{CW})}} =$
   $\overline{L_b(\mathcal{GEN}_A) \cap \overline{L_b(\mathcal{SPEC}_{TWC}) \cap L_b(\mathcal{SPEC}_{CW})}}$

2. $\overline{L_b(STWC) \cap \overline{L_b(\mathcal{GEN}_A) \cap L_b(\mathcal{SPEC}_{CW})}} =$
   $\overline{L_b(\mathcal{GEN}_A) \cap \overline{L_b(\mathcal{SPEC}_{TWC}) \cap L_b(\mathcal{SPEC}_{CW})}}$

3. $\overline{L_b(SCW) \cap \overline{L_b(\mathcal{GEN}_A) \cap L_b(\mathcal{SPEC}_{TWC})}} =$
   $\overline{L_b(\mathcal{GEN}_A) \cap \overline{L_b(\mathcal{GEN}_A) \cap L_b(\mathcal{SPEC}_{TWC})}}$

En effectuant ces tests avec la commande **NONCONFLICT** de TCT, nous obtenons les résultats suivants:

1. **NONCONFLICT**($\mathcal{GEN}_A$, $TWCCW_m$)=**TRUE**

2. **NONCONFLICT**($STWC$, GCW_m) = **FALSE**

3. **NONCONFLICT**(SCW, $GTWC_m$)=**FALSE**

De ces résultats nous pouvons déduire que si l'un de ces deux services est implanté au commutateur, l'ajout de l'autre service va provoquer une interférence avec le service déjà existant. Ce résultat n'est pas étonnant vu que chacun de ces deux services interprète de façon différente l'occurrence de l'événement $flash_A$. Nous illustrons dans la section 5.2 des exemples de mots qui créent des interférences.

5.1.2 Test d'interférence entre les services "conférence à trois" et "transfert d'appel, ligne occupée"

Soit $TWCCFBL_m$ la conjonction des deux superviseurs $STWC_m$ et $SCFBL_m$. Cet automate peut être obtenu en exécutant la commande

$$\mathbf{TWCCFBL_m = MEET(STWC_m, SCFBL_m)}$$

de TCT. Étant donné que $\mathcal{SPEC}_{TWC}$ et $\mathcal{SPEC}_{CFBL}$ sont co-accessibles, alors:

$$L_b(TWCCFBL_m) = \overline{L_b(\mathcal{SPEC}_{TWC})} \cap \overline{L_b(\mathcal{SPEC}_{CFBL})}.$$

Étant donné que $L_b(\mathcal{SPEC}_{TWC})$ et $L_b(\mathcal{SPEC}_{CFBL})$ sont non conflictuels

$$\mathbf{NONCONFLICT}(\mathcal{SPEC}_{TWC}, \mathcal{SPEC}_{CFBL}) = \mathbf{TRUE}$$

Alors nous pouvons déduire d'après la définition 3.9 que:

$$\overline{L_b(\mathcal{SPEC}_{TWC})} \cap \overline{L_b(\mathcal{SPEC}_{CFBL})} = \overline{L_b(\mathcal{SPEC}_{TWC}) \cap L_b(\mathcal{SPEC}_{CFBL})}$$

et par conséquent

$$L_b(TWCCFBL_m) = \overline{L_b(\mathcal{SPEC}_{TWC}) \cap L_b(\mathcal{SPEC}_{CFBL})}.$$

Soit $GCFBL_m$ l'automate qui modélise le fonctionnement en boucle fermée du

générateur GEN_AM sous la supervision de $SCFBL_m$:

$$\mathbf{GCFBL_m = MEET(SCFBL_m, GEN_AM).}$$

Remarquons que puisque $L_b(\mathcal{SPEC}_{CFBL})$ et $L_b(\mathcal{GEN}_A)$ sont non conflictuels, alors

$$\begin{aligned} L_b(GCFBL1_m) &= \overline{L_b(\mathcal{SPEC}_{CFBL}/GEN_A)} \\ &= \overline{L_b(\mathcal{GEN}_A) \cap L_b(\mathcal{SPEC}_{CFBL})} \\ &= \overline{L_b(\mathcal{GEN}_A)} \cap \overline{L_b(\mathcal{SPEC}_{CFBL})}. \end{aligned}$$

Pour voir si les deux services "conférence à trois" et "transfert d'appel, ligne occupée" sont interférents, nous effectuons les tests de non conflictualité suivants:

1. $\overline{L_b(\mathcal{GEN}_A)} \cap \overline{L_b(\mathcal{SPEC}_{TWC}) \cap L_b(\mathcal{SPEC}_{CFBL})} =$
   $\overline{L_b(\mathcal{GEN}_A) \cap \overline{L_b(\mathcal{SPEC}_{TWC}) \cap L_b(\mathcal{SPEC}_{CFBL})}}$
2. $\overline{L_b(STWC)} \cap \overline{L_b(\mathcal{GEN}_A) \cap L_b(\mathcal{SPEC}_{CFBL})} =$
   $\overline{L_b(\mathcal{GEN}_A) \cap \overline{L_b(\mathcal{SPEC}_{TWC}) \cap L_b(\mathcal{SPEC}_{CFBL})}}$
3. $\overline{L_b(SCFBL)} \cap \overline{L_b(\mathcal{GEN}_A) \cap L_b(\mathcal{SPEC}_{TWC})} =$
   $\overline{L_b(SCFBL) \cap \overline{L_b(\mathcal{GEN}_A) \cap L_b(\mathcal{SPEC}_{TWC})}}$

En effectuant ces tests avec la commande **NONCONFLICT** de TCT, nous obtenons les résultats suivants:

1. **NONCONFLICT**($\mathcal{GEN}_A$, $TWCCFBL_m$)=**TRUE**
2. **NONCONFLICT**($STWC$, $GCFBL_m$) = **FALSE**
3. **NONCONFLICT**($SCFBL$, $GTWC_m$)=**FALSE**

Donc, les deux services TWC et CFBL sont interférents.

5.1.3 Test d'interférence entre les services "appel en attente" et" transfert d'appel, ligne occupée"

Comme c'était le cas dans les sections 5.1.1 et 5.1.2, nous avons à effectuer les tests de non conflictualité suivants:

1. $\overline{L_b(\mathcal{GEN}_A)} \cap \overline{L_b(\mathcal{SPEC}_{CW}) \cap L_b(\mathcal{SPEC}_{CFBL})} = \overline{L_b(\mathcal{GEN}_A) \cap \overline{L_b(\mathcal{SPEC}_{CW}) \cap L_b(\mathcal{SPEC}_{CFBL})}}$

2. $\overline{L_b(SCW)} \cap \overline{L_b(\mathcal{GEN}_A) \cap L_b(\mathcal{SPEC}_{CFBL})} = \overline{L_b(\mathcal{GEN}_A) \cap \overline{L_b(\mathcal{SPEC}_{CW}) \cap L_b(\mathcal{SPEC}_{CFBL})}}$

3. $\overline{L_b(SCFBL)} \cap \overline{L_b(\mathcal{GEN}_A) \cap L_b(\mathcal{SPEC}_{CW})} = \overline{L_b(SCFBL) \cap \overline{L_b(\mathcal{GEN}_A) \cap L_b(\mathcal{SPEC}_{CW})}}$

En effectuant ces tests avec la commande **NONCONFLICT** de TCT, nous obtenons les résultats suivants:

1. **NONCONFLICT**($\mathcal{GEN}_A$, $CWCFBL_m$)=**TRUE**

2. **NONCONFLICT**(SCW, $GCFBL_m$) = **FALSE**

3. **NONCONFLICT**($SCFBL$, GCW_m)=**FALSE**

Où $CWCFBL_m$ est la conjonction des deux superviseurs CW_m et $SCFBL_m$.

Donc, si nous considérons le fonctionnement du commutateur sous la supervision de l'un de ces deux services, ces résultats nous permettent de déduire que l'implantation de l'autre service va provoquer une interférence avec le service existant à cause de la différence dans l'interprétation de l'occurrence de l'événement $flash_A$.

5.1.4 Test d'interférence entre les trois services

Dans les sections 5.1.1, 5.1.2 et 5.1.3 nous avons testé les interférences des services par groupes de deux. Dans cette section, nous allons tester l'interférence des trois services implantés ensemble. Pour cela nous devons tout d'abord calculer les automates suivants:

1. **MEET**($TWCCW_m$,$SCFBL_m$)=$TWCCWCFBL_m$
2. **MEET**(GCW_m,$SCFBL_m$)=$GCWCFBL_m$
3. **MEET**($GTWC_m$,SCW_m)=$GTWCCW_m$
4. **MEET**($GTWC_m$,$SCFBL_m$)=$GTWCCFBL_m$

Une fois que ces automates sont calculés nous effectuons les tests de non conflictualité. Les résultats obtenus sont les suivants:

1. **NONCONFLICT**($\mathcal{GEN}_A$,$TWCCWCFBL_m$)=**TRUE**
2. **NONCONFLICT**($STWC$,$GCWCFBL_m$)=**FALSE**
3. **NONCONFLICT**(SCW,$GTWCCFBL_m$)=**FALSE**
4. **NONCONFLICT**($SCFBL$,$GTWCCW_m$)=**FALSE**

Donc il est clair que l'implantation de ces trois services à un commutateur produit des interférences entre elles.

5.2 Discussion des résultats

Nous avons montré à la section précédente que les trois services provoquent un conflit lorsqu'ils sont implantés simultanément dans un même commutateur local. Nous

allons maintenant justifier ces résultats un par un. Considérons les deux services TWC et CW. Ces deux services permettent une communication d'un abonné A avec deux autres abonnés B et C en même temps sauf que le service TWC exige que l'abonné A initie les deux appels (con_{Ai}), alors que dans le service CW l'abonné A initie le premier appel par con_{Ai}, mais pas le deuxième qui est initié par un autre abonné avec un con_{iA}. Cependant, les deux services sont déclenchés par le même événement $flash_A$. En termes de langages formels ceci implique qu'un mot $m \in \overline{L(\mathcal{SPEC}_{TWC})} \cap \overline{L(\mathcal{SPEC}_{CW})}$ n'appartient pas forcément à $\overline{L(\mathcal{SPEC}_{TWC}) \cap L(\mathcal{SPEC}_{CW})}$. En effet, si m est une suite d'événements qui se termine par un $flash_A$, elle peut être étendue soit par un con_{iA} dans le cas du langage légal de $\mathcal{SPEC}_{TWC}$ et dans ce cas m ne pourra plus être étendu en un mot de $L(\mathcal{SPEC}_{CW})$, soit par un con_{Ai} dans le cas du langage légal de $\mathcal{SPEC}_{CW}$ et dans ce cas m ne pourra plus être étendu en un mot de $L(\mathcal{SPEC}_{TWC})$. Donc la différence de l'interprétation du $flash_A$ est à l'origine du conflit entre les deux services TWC et CW.

Considérons maintenant les deux services CW et CFBL. Selon les spécifications que nous avons développées ces deux services sont en conflit lorsqu'ils sont implantés simultanément au niveau du même commutateur local A. Le service CW permet de prendre un deuxième appel lorsque l'abonné est déjà en communication, alors que le service CFBL permet d'acheminer les appels d'un deuxième appelant lorsque l'abonné est en communication. Il est clair de cet énoncé que les deux services sont en conflit. En effet, si A est abonné aux deux services en même temps et qu'il est en communication avec un autre abonné, nous ne pouvons pas déterminer à priori lequel des deux services doit être activé. Est-ce le service CW qui va informer l'abonné A de l'arrivée d'un deuxième appel et donc l'événement $flash_A$ va mettre A avec le nouveau appelant, ou est-ce le service CFBL qui va acheminer ce nouvel appel à un autre abonné

étant donné que A est déjà en communication? Ces deux services sont déclenchés par le même événement $dial_{iA}$. Cependant, cet événement est interprété différement par les deux services. Pour le langage légal de la spécification du service CW ce $dial_{iA}$ doit être suivi par un $flash_A$ et un con_{iA} pour former un mot marqué, alors que pour le langage légal de la spécification du service CFBL ce $dial_{iA}$ doit être suivi par un con_{iAj} pour former un mot marqué. Donc la différence dans l'interprétation de l'événement $dial_{iA}$ est à l'origine du conflit entre les deux services CW et CFBL.

Le conflit entre les deux services TWC et CFBL est un peu différent. En fait, dans ce cas, il n' y a pas vraiment un même événement responsable du déclenchement des deux services comme c'était le cas pour les autres conflits étudiés précédement. Analysons les états 2 et 3 de chacun des deux automates dévéloppés pour les services TWC et CFBL. Étant donné que les états 2 et 3 jouent des rôles similaires, nous allons nous limiter dans notre analyse à l'état 2. Dans les deux automates l'état 2 correspond au cas où l'abonné A est en communication avec l'abonné B. Rappelons aussi que les états qui désignent des tâches complètes dans $\mathcal{SPEC}_{TWC}$ et $\mathcal{SPEC}_{CFBL}$ sont les états 4 et 5. À partir de l'état 2, l'automate $\mathcal{SPEC}_{TWC}$ peut aboutir à l'état 4 en l'occurrence de la suite d'événements $flash_A.dial_{AC}.con_{AC}$, en plus l'occurrence de l'événement $dial_{CA}$ ne peut être suivie que par un $nocon_{CA}$ puisque l'événement con_{CAB} n'est pas permis dans la spécification de ce service. Ceci n'est plus vrai dans le cas de l'automate du service CFBL. En effet, à partir de l'état 2, l'automate $\mathcal{SPEC}_{CFBL}$ peut aboutir à l'état 4 avec l'occurrence de la suite d'événements $dial_{CA}.con_{CAB}$, en plus à partir de cet état l'événement $dial_{AC}$ n'est pas permis après l'occurrence d'un $flash_A$ comme c'était le cas dans l'automate $\mathcal{SPEC}_{TWC}$. En conclusion, ni l'occurrence de l'événement $dial_{CA}$, ni l'occurrence de l'événement $dial_{AC}$ permet aux deux automates d'aboutir à un état achevé simultanément. Rappelons que selon notre

modèle, il n'y a conflit entre ces deux services que dans le cas où ils sont implantés tous les deux dans le même commutateur.

Nous avons montré aussi que la présence simultanée des trois services engendre un conflit entre eux. Ceci est tout à fait normal étant donné que chaque couple de ces services provoque une interférence.

Quoique les trois interférences détectées semblent être dues au même problème, nous pensons qu'il y a quelques différences entre elles. Revenons au conflit entre les services TWC et CW. Il est clair que l'interférence est due à l'activation des deux services par l'occurrence du même événement $flash_A$ qui est interprété différement par les deux services, sauf que dans ce cas de conflit, nous pouvons déterminer lequel des deux services doit être activé à partir de la suite d'événements engendrée par le système. En effet, si $flash_A$ est engendré après l'occurrence d'un $dial_{iA}$ il est clair que $flash_A$ est engendré pour prendre un "appel en attente", et par conséquent c'est le service CW qui doit être activé et pas TWC. Dans ce cas, le problème revient donc à attribuer une priorité plus élévée au service CW. Par contre, si l'événement $flash_A$ est engendré sans qu'il n'y ait eu au préalable une occurrence d'un événement $dial_{iA}$, il est évident que c'est le service TWC qui doit être activé, et dans ce cas ce service doit être plus prioritaire que le service CW. Donc l'attribution de la priorité n'est pas statique et elle dépend de la suite d'événements engendrés par le système.

Regardons maintenant le conflit entre les deux services CW et CFBL. Comme c'était le cas pour les deux services TWC et CW, le conflit entre CW et CFBL est dû au déclenchement des deux services par le même événement. Comme expliqué précédemment l'événement $dial_{iA}$ active les deux services et par conséquent il est interprété de deux manières différentes, ce qui cause le problème d'interférence. Cependant, la suite des événements engendrés ne nous donne aucune information sur le

service qui doit être prioritaire et donc activé dans le cas de l'occurrence d'un $dial_{iA}$. Dans ce cas-ci il s'agit d'attribuer une priorité fixe aux deux services, ce qui implique que nous devons choisir au préalable lequel des deux services sera prioritaire.

Nous avons déjà expliqué l'origine du conflit entre les deux services TWC et CFBL et nous avons vu que ce conflit est différent des deux autres cas puisque il n'est pas dû au déclenchement des deux services par le l'occurrence d'un même événement qui est interprété différement. Nous pensons que ce type de conflit ne doit pas être résolu par l'attribution d'une priorité entre les deux services, mais plutôt par l'enrichissement de la spécification d'un service pour qu'elle puisse supporter l'implantation de l'autre, sauf que ceci contredit notre choix de faire une synthèse modulaire, où chaque service est implanté indépendement des autres.

Comme nous l'avons noté dans le chapitre précédent, le petit modèle du système téléphonique que nous avons considéré ne contient que trois abonnés. En réalité, pour avoir une spécification du service CFBL conforme à la documentation fournie nous avons besoin d'un modèle qui contient au moins quatre abonnés. Le scénario du service devrait être le suivant : A est en communication avec un abonné B et C appelle A, mais puisque la ligne de ce dernier est occupée, alors l'appel est acheminé vers un autre abonné D. Pour éviter d'augmenter inutilement la taille du générateur et des spécifications, nous avons supposé que l'abonné avec lequel A est en communication initie le deuxième appel. Notre idée est que le conflit n'est pas dû dans notre cas au nombre d'abonnés impliqués dans les services mais plutôt à la différence de l'interprétation des événements par ces services.

Lorsque nous manipulons des automates de grande taille il est très difficile de faire l'étude de ces automates manuellement sans avoir un support logiciel. Étant donné que le seul outil disponible était TCT, nous avons décidé de faire notre étude à

l'aide de cet outil. Cependant, parmi les inconvénients majeurs de TCT, en plus des plusieurs bugs qu'il contient, est le fait qu'il ne permet pas l'étude des grands systèmes. Avec son interface non conviviale et son mode d'utilisation où les événements doivent être codés, la manipulation des automates devient une tâche pénible surtout lorsqu'il s'agit de corriger une erreur par exemple. Il était ainsi important de développer un générateur et des spécifications qui soient d'une part les plus simples possibles et d'autre part les plus réalistes possibles et qui traduisent fidèlement les réseaux téléphoniques réels. D'ici vient la première difficulté de notre travail. En s'inspirant des anciens travaux et des documentations fournies pour le concours, nous avons réussi à dévélopper un modèle du générateur et des spécifications assez simples pour étudier ces exemples d'interférences.

Chapitre 6

CONCLUSION

Le problème de conflit entre services est considéré maintenant l'un des problèmes les plus prioritaires par les administrateurs des réseaux téléphoniques. Plusieurs compagnies et chercheurs dans le monde entier s'intéressent de plus en plus à ce phénomène et essaient de trouver des méthodes rigoureuses de détection et de résolution. D'après [Kim97], trouver une solution à ce problème est plus important que fournir de nouveaux services. Ceci est tout à fait normal avec les exigences croissantes des clients qui préfèrent un nombre réduit de services qui fonctionnent correctement qu'un nombre élévé de services mais dont les comportements en commun sont indésirables.

Dans ce mémoire nous avons pris trois exemples de services: TWC, CW et CFBL. Les spécifications se sont inspirées du document fourni pour le concours qui a eu lieu dans le cadre du cinquième atelier international sur les intéractions de services (5th IWFI). Dans un premier temps, Nous avons dévéloppé un petit modèle d'un réseau téléphonique constitué de trois abonnés. Ensuite, nous avons essayé de dévélopper des spécifications les plus simples possibles mais qui soient en même temps le plus proche de la réalité selon la documentation. Pendant la modélisation nous avons signalé

l'importance de chaque événement de l'alphabet local du commutateur téléphonique et le rôle de chaque état de l'automate et sa signification dans notre modèle. Il était donc nécessaire de savoir quels étaient les événements importants dans la synthèse des automates des spécifications de services qui provoquent un changement dans l'état du système d'une part, et qui caractérisent le service étudié d'autre part.

Les langages légaux des spécifications développées possèdent des propriétés particulières. Ils sont tous co-accessibles, commandables et non conflictuels par rapport au langage engendré par le générateur du système. La vérification de ces propriétés signifient que chacune des spécifications développées implante un superviseur non bloquant par rapport au générateur.

Après ceci nous avons effectué des tests pour chercher s'il y a interférences entre ces services, et nous avons détecté un conflit entre chaque paire. Nous avons analysé ces résultats et montré l'origine de chaque conflit. Nous avons signalé aussi une méthode de résolution de ces interférences basée sur une attribution de priorités entre services.

La méthode que nous avons utilisée pour la détection de conflit entre services téléphoniques est une méthode formelle basée sur la théorie des systèmes à événements discrets, et comme toute autre méthode formelle, elle offre des solutions indépendantes du type du réseau téléphonique. Ceci implique que l'étude faite peut être adoptée pour n'importe quelle architecture du système. En plus, notre analyse s'effectue à l'étape de conception des services se qui assure la non propagation du problème de conflit à l'étape d'implémentation.

La différence la plus évidente entre notre approche et les autres approches au problème d'interférence est que la nôtre tient compte d'une notion de commandabilité, ou plus précisément, elle tient compte de la répartition de la commande entre différents services. Une conséquence de la prise en compte explicite de la commandabilité

est la possibilité de faire une synthèse formelle de superviseurs dans des cas où la spécification est non commandable. (voir [Mun98] pour un exemple de synthèse d'un superviseur pour une spécification non commandable). Cette synthèse revient essentiellement au calcul du plus grand sous-langage commandable (voir la proposition 3.3).

Pour réduire la complexité du problème, la théorie de la commande des systèmes à événements discrets offre la possibilité de faire une synthèse modulaire. Ainsi, nous avons utilisé une synthèse modulaire décentralisée basée sur la décomposition du problème de synthèse en sous problèmes plus faciles à résoudre, où chaque spécification est modélisée indépendamment des autres. Dans [TMH97, TMHL97] nous trouvons d'autres exemples de synthèse modulaire du problème de conflit, ainsi qu'une approche de résolution de conflit basée sur la commande hiérarchisée des systèmes à événements discrets.

Durant l'étude de notre modèle de réseau téléphonique nous avons remarqué que l'observabilité des événements par rapport à un commutateur local n'est pas tout à fait une observabilité fixe, où un événement est soit observable soit non observable par un commutateur. Nous donnons comme exemple l'événement $offhook_B$ qui correspond à un décrochement du combiné de B et n'est pas toujours non observable par rapport à A. C'est vrai que si B décroche son combiné pour initialiser un appel cet événement n'est pas observable par A, mais si B décroche son combiné suite à l'occurrence de l'événement $dial_{AB}$, alors cet événement devient observable par A. L'événement $flash_B$ n'est pas observable par A, mais nous pensons que ce n'est plus le cas si A et B sont en communication. Il s'agit donc d'une observation partielle des événements, où l'observabilité d'un événement dépend de l'état du système, ou plutôt du mot engendré par l'automate.

Bien que notre modèle de trois abonnés s'avère suffisant pour étudier les services TWC, CW et CFBL et détecter ses interférences, il est évident que plusieurs services nécéssitent un plus grand nombre d'abonnés pour pouvoir détecter la présence des conflits, comme nous l'avons mentionné pour le service CFBL. Cependant, le seul outil disponible jusqu'à date est le logiciel TCT qui est incapable de manipuler de grands systèmes. Il est donc très important de dévélopper un autre outil qui soit plus convivial, plus facile à utiliser et qui permet l'étude de systèmes plus grands et plus réalistes.

Le travail que nous avons réalisé est une première étape de recherche qui ouvre la voie à la synthèse de systèmes temps réel plus élaborés. Nous pensons en particulier à la temporisation des automates, qui permettera l'étude de plusieurs autres problèmes comme ceux liés à la facturation. Pour réduire le nombre de tests d'interférence à effectuer, nous pouvons aussi introduire un mécanisme de filtrage qui élimine toute les combinaisons de services pour lesquels nous savons d'avance qu'ils n'interfèrent pas. Dans [TMH97, TMHL97, WTHM97] nous trouvons une approche hiérarchisée de synthèse et de résolution de conflit (voir section 2.6). Il serait important d'enrichir ce travail par une telle approche pour la résolution de conflit entre les services développés.

Bibliographie

[AC97] Imene Aggoun and Pierre Combes. Observers in the sce and the see to detect and resolve service interactions. Dans Petre Dini, Raouf Boutaba, and Luigi Logrippo, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems IV*, pages 198–212. IOS Press, 1997.

[BEM92] N. Boudrigua, F. Elloumi, and A. Mili. On the lattice of specifications: Application to a specification methodology. Dans *Formal Aspects of Computing*, pages 544–571, 1992.

[CLL96] Yi-Liang Chen, Stéphane Laforture, and Feng Lin. Priority assignment algorithms for resolving blocking in modular control of discrete event systems. Dans *35th IEEE Conference on Decision and Control*, pages 2743–2748, déc. 1996.

[CLL97] Yi-Liang Chen, Stéphane Laforture, and Feng Lin. Resolving feature interactions using modular supervisory control with priorities. Dans Petre Dini, Raouf Boutaba, and Luigi Logrippo, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems IV*, pages 108–122. IOS Press, 1997.

[DBS+95] J. Desharnais, N. Belkhiter, S. Ben Mohamed Sghaier, F. Tchier, A. Jaoua, A. Mili, and N. Zaguia. Embedding a demonic semilattice in a relation algebra. Dans *Theoretical Computer Science*, pages 333–360, 1995.

[FMD97] M. Frappier, A. Mili, and J. Desharnais. Detecting feature interactions on relational specifications. Dans Petre Dini, Raouf Boutaba, and Luigi Logrippo, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems IV*, pages 123–137. IOS Press, 1997.

[GBGO98] Nancy Griffeth, Ralph Blumenthal, Jean-Charles Gregoire, and Tadashi Ohta. Feature interaction detection contest. Dans K. Kimbler and L.G. Bouma, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems V*, pages 327–359. IOS Press, 1998.

[Gib97] J. Paul Gibson. Feature interaction models: Understanding interactions. Dans Petre Dini, Raouf Boutaba, and Luigi Logrippo, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems IV*, pages 46–60. IOS Press, 1997.

[Gib98] J. Paul Gibson. Towards a feature interaction algebra. Dans K. Kimbler and L.G. Bouma, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems V*, pages 217–231. IOS Press, 1998.

[HU69] J.E. Hopcroft and J.D. Ullman. *Formal Languages and Their Relation to Automata.* Addison-Wesley Publishing Company, USA, 1969.

[IE97] Y. Iraqi and M. Erradi. An experiment for the processing of feature interactions within an object-oriented environment. Dans Petre Dini,

Raouf Boutaba, and Luigi Logrippo, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems IV*, pages 298–312. IOS Press, 1997.

[Joh97] Blom Johan. Formalisation of requirements with emphasis on feature interaction detection. Dans Petre Dini, Raouf Boutaba, and Luigi Logrippo, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems IV*, pages 61–77. IOS Press, 1997.

[JZ96] Michael Jackson and Pamela Zave. Where do operations come from: A multiparadigm specification technique. *IEEE Transactions on Software Engineering*, 22(7):508–528, 1996.

[JZ98] Michael Jackson and Pamela Zave. Distributed feature composition: A virtual architecture for telecommunications services. *IEEE Transactions on Software Engineering*, 24(10):831–847, 1998.

[Kim97] Kristofer Kimbler. Addressing the interaction problem at the entreprise level. Dans Petre Dini, Raouf Boutaba, and Luigi Logrippo, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems IV*, pages 13–22. IOS Press, 1997.

[KL98] Jalel Kamoun and Luigi Logrippo. Goal-oriented feature interaction detection in the intelligent network model. Dans K. Kimbler and L.G Bouma, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems V*, pages 172–186. IOS Press, 1998.

[Mun98] A. Munteanu. L'analyse des services téléphoniques dans le cadre de la théorie de la commande des systèmes à événements discrets. Mémoire de

maîtrise, département de génie éléctrique et de génie informatique, École Polytechnique de Montréal, Canada, 1998.

[Pre97] Christian Prehofer. An object-oriented approach to feature interaction. Dans Petre Dini, Raouf Boutaba, and Luigi Logrippo, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems IV*, pages 313–325. IOS Press, 1997.

[RW89] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. Dans *Proceedings of the IEEE*, pages 81–98, 1989.

[Thi96] John G. Thistle. Supervisory control of discret-event systems. *Mathematical and Computer Modelling*, 23:25–53, 1996.

[THM93] John G. Thistle, H.-H. Hoang, and R. P. Malhamé. Modélisation et analyse du traitement des appels téléphoniques par l'approche de ramadge-wonham: Rapport préliminaire. Technical Report 93/15, École Polytechnique de Montréal, Canada, Août 1993.

[Tho97] Muffy Thomas. Modeling and analysing user views of telecommunications services. Dans Petre Dini, Raouf Boutaba, and Luigi Logrippo, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems IV*, pages 168–182. IOS Press, 1997.

[TMH97] John G. Thistle, R. P. Malhamé, and H.-H. Hoang. Feature interaction modelling detection and resolution: A supervisory control approach. Dans Petre Dini, Raouf Boutaba, and Luigi Logrippo, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems IV*, pages 93–107. IOS Press, 1997.

[TMHL97] John G. Thistle, R. P. Malhamé, H.-H. Hoang, and S. L. Lafortune. Supervisory control of distributed systems, Part I: Modelling, specification and synthesis. Technical Report 97/08, École Polytechnique de Montréal, Canada, Fevrier 1997.

[Tur97] Kenneth J. Turner. An architectural foundation for relating features. Dans Petre Dini, Raouf Boutaba, and Luigi Logrippo, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems IV*, pages 226–241. IOS Press, 1997.

[Tur98] Kenneth J. Turner. Validating architectural feature descriptions using lotos. Dans K. Kimbler and L.G. Bouma, rédacteurs, *Feature Interactions in Telecommunications and Distributed Systems V*, pages 247–261. IOS Press, 1998.

[Won88] W.M. Wonham. A control theory for discrete-event-systems. *Advanced Computing Concepts and Techniques in Control Engineering*, F47:129–169, 1988.

[WTHM95] K.C. Wong, J.G. Thistle, H-H. Hoang, and R.P. Malhamé. Conflict resolution with flexible priority in modular control with application to feature interaction. Dans *35th IEEE Conference on Decision and Control*, pages 415–421, déc. 1995.

[WTHM97] K. C. Wong, J. G. Thistle, H.-H. Hoang, and R. P. Malhamé. Supervisory control of distributed systems, Part II: Conflict resolution. Technical Report 97/08, École Polytechnique de Montréal, Canada, Fevrier 1997.

[Zav98] Pamela Zave. Systematic design of call-coverage features. *Algebraic Methodology and Software Technology*, pages 23–27, 1998.

[ZW90] H. Zhong and W.M. Wonham. On the consistency of hierarchical supervision in discrete event systems. *IEEE Transactions in Automatic Control*, 35(10):1125–1134, 1990.

Annexe A

Les fichiers TCT

Les automates définis dans le présent rapport ont été dévéloppés avec TCT sous MS-DOS. Chaque automate est défini dans un fichier séparé qui porte le même nom avec ".des" comme extension. Notons que ces fichiers peuvent être manipulés avec la version de TCT sous windows (tctw.exe) sans aucun problème. Il est peut-être utile de noter – d'après notre expérience avec cet outil – que la version TCT sous MS-DOS reconnaît les noms de fichiers qui ne dépassent pas huit caractères (sans compter l'extension), alors que la version sous windows ne reconnaît que les noms d'au plus sept caractères. Pour cette raison nous avons pris attention pour que les noms de nos fichiers ne dépassent pas sept caractères pour qu'ils puissent être manipulés sur les deux versions.

Nous listons dans le tableau A.1 les noms des fichiers créés avec TCT pour tous les automates dévéloppés. Dans la première colonne nous donnons le nom de fichier. Dans la deuxième, le nom de l'automate tel que donné dans le présent mémoire. Dans la troisième, la page et la section où l'automate a été défini pour la première fois.

Fichier TCT (.des)	**Automate**	**Page : Section**
SWITCHA	$\mathcal{SWITCH}_A$	44 : 4.1.2
SWITCHB	$\mathcal{SWITCH}_B$	46 : 4.1.3
SWITCHC	$\mathcal{SWITCH}_C$	46 : 4.1.3
RESEAU	$\mathcal{GLOBAL}$	46 : 4.1.3
LOCAL	$\mathcal{LOCAL}_A$	46 : 4.1.3
FLASHA	$\mathcal{FLASH}_A$	47 : 4.2
GENA	$\mathcal{GEN}_A$	49 : 4.2
SPECTWC	$\mathcal{SPEC}_{TWC}$	50 : 4.3
SPECCW	$\mathcal{SPEC}_{CW}$	53 : 4.4
SPECCF	$\mathcal{SPEC}_{CFBL}$	56 : 4.5
GENAM	GEN_AM	62 : 4.6
STWC	$STWC$	61: 4.6
SCW	SCW	61 : 4.6
SCFBL	$SCFBL2$	61 : 4.6
STWCM	$STWCM$	64 : 5.1
SCWM	$SCWM$	64 : 5.1
SCFBLM	$SCFBLM$	64 : 5.1
TWCCWM	$TWCCW_m$	64 : 5.1.1
GTWCM	$GTWC_m$	65 : 5.1.1
GCWM	GCW_m	65 : 5.1.1
TWCCFM	$TWCCFBL_m$	67 : 5.1.2
CWCFM	$CWCFBL_m$	69 : 5.1.3
GCFM	$GCFBL_m$	68 : 5.1.2
S123M	$TWCCWCFBL_m$	70 : 5.1.4
G23M	$GCWCFBL_m$	69 : 5.1.3
G12M	$GTWCCW_m$	70 : 5.1.4
G13M	$GTWCCFBL2_m$	70 : 5.1.4

Tableau A.1: Liste des fichiers TCT.